



महाराष्ट्र राज्य तंत्रशिक्षण मंडळ, मुंबई
(स्वायत्व) (ISO 9001:2015) (ISO/IEC 27001:2013)

अभियांत्रिकी आणि तंत्रज्ञान पदविका

शिक्षण पुस्तिका
(Learning Material)

DATA STRUCTURE USING PYTHON

(313306)

K Scheme

डेटा स्ट्रक्चर युजिंग पायथन

आर्टिफिशियल इंटेलिजन्स अभियांत्रिकी गट
(के स्कीम)

मराठी-इंग्रजी (द्विभाषिक) माध्यम
(अभियांत्रिकी व तंत्रज्ञानातील तिसरे सत्र पदविका)

शिक्षण पुस्तिका
(Learning Material)

डेटा स्ट्रक्चर युजिंग पायथन

DATA STRUCTURE USING PYTHON

(313306)

K Scheme

आर्टिफिशियल इंटेलिजन्स अभियांत्रिकी गट
मराठी-इंग्रजी (द्विभाषिक) माध्यम
(अभियांत्रिकी व तंत्रज्ञानातील तृतीय सत्र पदविका)



महाराष्ट्र राज्य तंत्रशिक्षण मंडळ, मुंबई
(स्वायत्व) (ISO 9001:2015) (ISO/IEC 27001:2013)

(313306)

मार्गदर्शक

प्रा. कुकरे विजय नामदेवराव

विभागप्रमुख, संगणक अभियांत्रिकी

प्रा. काळे गोरक्षनाथ भागवतराव

उपप्राचार्य व विभागप्रमुख, संगणक तंत्रज्ञान

लेखक

प्रा. कडलग एस.यु.

(अधिव्याख्याता, माहिती तंत्रज्ञान विभाग)

प्रा. कासार वाय.एस.

(अधिव्याख्याता, माहिती तंत्रज्ञान विभाग)

प्रा. बाहेती एस.एस.

(अधिव्याख्याता, संगणक विभाग)

प्रा. बोन्हाडे एस.एस.

(अधिव्याख्याता, संगणक विभाग)

प्रा. वाकचौरे एस. एल.

(अधिव्याख्याता, संगणक विभाग)

प्रा. शिंदे बी.बी.

(अधिव्याख्याता, संगणक विभाग)



महाराष्ट्र राज्य तंत्र शिक्षण मंडळ

(स्वायत्त) (ISO: ९००१:२०१५) (ISO/IES: २७००१-२०१३)

शासकीय तंत्रनिकेतन इमारत, चौथा मजला, ४९, खेरवाडी, बांद्रा (पूर्व), मुंबई - ४०० ०५१.

दूरध्वनी क्र.: ०२२-६२५४२१७०/१६१

Email : director@msbte.com

Web : www.msbte.org.in



प्रास्ताविक

महाराष्ट्र राज्यातील पदविका स्तरावरील तंत्रशिक्षणामध्ये विद्यार्थ्यांचे रोजगार कौशल्य विकसित करून विद्यार्थ्यांचा सर्वांगीण विकास घडवून आणण्याकरिता महाराष्ट्र राज्य तंत्रशिक्षण मंडळ कटिबद्ध आहे. उद्योगधंद्यातील बदलत्या तंत्रज्ञानाशी संबंधित गरजा लक्षात घेऊन महाराष्ट्र राज्य तंत्र शिक्षण मंडळाकडून पदविका अभ्यासक्रम वेळोवेळी अद्यावत करण्यात येतो. अभियांत्रिकी पदविका अभ्यासक्रम शिकत असतांना संकल्पनात्मक ज्ञान, सुसंगत संदर्भ, प्रश्न विचारणे, विश्वसनीय पुरावे, कारणमीमांसा आणि सुस्पष्ट निकष यांचा वापर करून अर्थाची उकल करण्याची, विश्लेषण व मूल्यमापन करण्याची तसेच तर्काने अनुमान काढण्याची क्षमता म्हणजेच चिकित्सक विचार विद्यार्थ्यांमध्ये अधिक दृढ होतील असा मला विश्वास आहे. जेव्हा विद्यार्थी ज्ञान मिळवण्याच्या माध्यमाशी पूर्णपणे परिचित आणि सोयीस्कर असतात, तेव्हा त्यांच्यासाठी वर्गातील चर्चेत भाग घेणे सोपे होते, संकल्पनात्मक व सैद्धांतिक बाबींचे आकलन परिपूर्ण होते, संज्ञानात्मक क्षमता सुधारते आणि त्यांचा आत्मविश्वास देखील वाढतो या सर्व गोष्टींचा विचार करून मंडळाकडून शैक्षणिक सामग्रीची निर्मिती करण्यात आलेली आहे. भारत देश हा खेड्यापाड्यातून विकसित झालेला देश असून ग्रामीण भागातील विद्यार्थ्यांना तांत्रिक शिक्षण घेतांना भाषेचा अडसर न येता तांत्रिक बाबींचा आशय समजून घेणे शक्य होईल या दृष्टिकोनातून महाराष्ट्र राज्य तंत्र शिक्षण मंडळाने पदविका स्तरावरील तांत्रिक शिक्षणाकरिता विद्यार्थ्यांना मराठी-इंग्रजी द्विभाषिक माध्यमाचा पर्याय शैक्षणिक वर्ष २०२१-२२ पासून उपलब्ध करून दिलेला आहे.

राष्ट्रीय शैक्षणिक धोरण-२०२० प्रादेशिक भाषेतील शिक्षणास प्रोत्साहन देते, ज्यामुळे विद्यार्थ्यांना तांत्रिक अभ्यासक्रमांसाठी प्रादेशिक भाषांतुन शिक्षणाचे माध्यम निवडता येते. सदर धोरणामुळे प्रादेशिक भाषांमध्ये तांत्रिक सामग्री आणि अभ्यास सामग्रीचा विकास आणि भाषांतर निर्माण करण्याची आवश्यकता आहे. त्यास अनुसरून मंडळाने मराठी-इंग्रजी द्विभाषिक माध्यमाचा पर्याय द्वितीय व तृतीय वर्षाकरिताही उपलब्ध करून देण्यात आला आहे. तसेच त्याकरिताची शैक्षणिक सामग्रीही संबंधीत भागधारकरांना उपलब्ध करून देण्यात येत आहे.

पदविका स्तरावरील तंत्रशिक्षण अधिक दर्जेदार करण्यासाठी महाराष्ट्रातील अनुभवी व तज्ञ अध्यापकांनी व्यवहारिक मराठी भाषा व इंग्रजी भाषेतील तांत्रिक शब्दावली यांचा वापर करून मराठी - इंग्रजी भाषेचा सुवर्णमध्य साधण्याचा प्रयत्न केलेला आहे. मंडळाच्या स्तरावर गठीत सुकाणू समितीमार्फत सदर शैक्षणिक सामग्रीचा दर्जा, तसेच इतर बाबींची तपासणी करण्यात आलेली आहे. त्यामुळे सदर शैक्षणिक सामग्री अधिक सम्पन्न झालेली असून विद्यार्थी त्यांच्या व्यक्तिमत्त्वाचा सुसंवादी आणि सर्वांगीण विकास साधतील. परिणामतः विश्वस्तरीय मनुष्यबळाच्या गरजा पूर्ण करण्यात महाराष्ट्र राज्य अग्रेसर राहील व पर्यायाने राष्ट्रनिर्मिती करीता निश्चितच हातभार लागेल, असा मला विश्वास आहे.

अभियांत्रिकी पदविका अभ्यासक्रमातील प्रमुख विषयांची मराठी-इंग्रजी द्विभाषिक शैक्षणिक सामग्री बनविण्यासाठी अध्यापक व सुकाणू समितीचे सदस्य यांनी दर्शविलेले समर्पण व वचनबद्धता कौतुकास पात्र आहे, या सर्वांचे मी मनः पूर्वक अभिनंदन करतो !


(प्रमोद नाईक)
संचालक

म. रा. तंत्र शिक्षण मंडळ, मुंबई.

अनुक्रमणिका

अनु	युनिटचे नाव	पृष्ठ क्रमांक
1.	पायथॉन परिचय आणि पायथॉन मधील नियंत्रण प्रवाह विधाने (Introduction and Control flow statements in python)	1
2.	पायथनमधील विशिष्ट डेटा स्ट्रक्चर्स आणि फंक्शन्स (Python Specific Data Structure and Functions)	18
3.	पायथॉन मॉड्युल्स आणि पॅकेज (Python, Modules and Packages)	39
4.	पायथन मध्ये ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (Object Oriented Programming in Python)	58
5.	लिनियर डेटा स्ट्रक्चर , अ‍ॅर, लिंक लिस्ट, स्टॅक आणि क्यूज युझिंग पायथन (Linear Data Structure Arrays, Link List, Stack and Queues using Python)	75
6.	नॉन-लिनियर डेटा स्ट्रक्चर (Non-Linear Data Structure)	108

युनिट -1

पायथॉन परिचय आणि पायथॉन मधील नियंत्रण प्रवाह विधाने

(Introduction and Control flow statements in Python)

विषय निष्पत्ती (Course Outcome): मूलभूत वाक्यरचना वापरून पायथन प्रोग्राम विकसित करा.

घटक निष्पत्ती (Theory Learning Outcomes):

1. पायथनमध्ये दिलेल्या व्हेरिएबल्स, कीवर्ड आणि स्थिरांकाचे वर्णन करा.
2. दिलेल्या प्रोग्राममध्ये इंडेंटेशन कमेंट वापरा.
3. अंकगणितीय अभिव्यक्ती लिहिण्यासाठी विविध प्रकारचे ऑपरेटर वापरा.
4. नियंत्रण प्रवाह वापरून पायथन प्रोग्राम लिहा.

1.1 पायथॉन ची वैशिष्ट्ये (Features of python):

- **इंटरॅक्टिव्ह (Interactive):**

मानक पायथन वितरण इंटरॅक्टिव्ह शेलसह येते जे आरईपीएलच्या तत्वावर कार्य करते (वाचा - मूल्यांकन - प्रिंट - लूप). शेल पायथन प्रोग्राम सादर करते. आपण कोणतीही वैध पायथन एक्सप्रेशन (expression) टाईप करू शकता आणि एंटर दाबू शकता. पायथन इंटरप्रेटर ताबडतोब प्रतिसाद परत करतो आणि प्रोग्राम पुढील एक्सप्रेशन (expression) वाचण्यासाठी परत येतो.

```
>>> 2*3+1
```

```
7
```

```
>>> print ("Hello World")
```

```
Hello World
```

लायब्ररीशी परिचित होण्यासाठी आणि त्याची कार्यक्षमता तपासण्यासाठी इंटरॅक्टिव्ह मोड विशेषतः उपयुक्त आहे.

- **ऑब्जेक्ट-ओरिएंटेड (Object oriented)**

पायथन ही पूर्णपणे ऑब्जेक्ट ओरिएंटेड भाषा आहे. पायथनचे डेटा मॉडेल असे आहे की, जरी आपण प्रोसेस ओरिएंटेड दृष्टीकोन अनुकूलित करू शकता, तरीही पायथन अंतर्गतपणे ऑब्जेक्ट-ओरिएंटेड कार्यपद्धती वापरतो. पायथन प्रोग्राममधील प्रत्येक गोष्ट एक ऑब्जेक्ट आहे. तथापि, पायथन सोयीस्कररित्या त्याच्या ऑब्जेक्ट ओरिएंटेशनला अनिवार्य किंवा प्रोसेस ओरिएंटेड भाषा म्हणून वापरण्यासाठी समाविष्ट करते - जसे की C प्रोग्रामिंग.

- **प्लॅटफॉर्म स्वतंत्र (Platform independent)**

पायथन ही क्रॉस-प्लॅटफॉर्म भाषा आहे. विंडोज, लिनक्स, मॅक ओएस, अँड्रॉइड ओएस अशा विविध ऑपरेटिंग सिस्टिम प्लॅटफॉर्मवर प्री-संकलित बायनरी वापरासाठी उपलब्ध आहेत.

पायथन प्रोग्राम प्रथम मध्यवर्ती प्लॅटफॉर्म स्वतंत्र बाइट कोडमध्ये संकलित केला जातो. त्यानंतर दुभाषियाच्या आतील आभासी मशीन बाइट कोड कार्यान्वित करते. हे वर्तन पायथनला क्रॉस-प्लॅटफॉर्म भाषा बनवते आणि अशा प्रकारे पायथन प्रोग्राम सहजपणे एका ओएस प्लॅटफॉर्मवरून दुसऱ्या ओएस प्लॅटफॉर्मवर पोर्ट केला जाऊ शकतो.

• इंटरप्रेटेड (Interpreted)

पायथन ही दुभाषिया-आधारित भाषा आहे. पायथन कोड एका वेळी एका विधानाद्वारे अंमलात आणला जातो. पायथन इंटरप्रेटरचे दोन घटक आहेत. अनुवादक वाक्यरचनेसाठी विधान तपासतो. योग्य आढळल्यास ते मध्यवर्ती बाइट कोड तयार करते. एक पायथन व्हर्च्युअल मशीन आहे जे नंतर मूळ बायनरीमध्ये बाइट कोड रूपांतरित करते आणि ते कार्यान्वित करते. दुभाषी एका वेळी सोर्स कोडमधून एक सूचना घेतो, त्याचे मशीन कोडमध्ये भाषांतर करतो आणि त्याची अंमलबजावणी करतो. त्रुटी च्या पहिल्या घटनेपूर्वीच्या सूचनांची अंमलबजावणी केली जाते. या वैशिष्ट्यासह, प्रोग्राम डिबग करणे सोपे आहे आणि अशा प्रकारे सुरुवातीच्या पातळीवरील प्रोग्रामरला हळूहळू कॉन्फिडेंसमिळविण्यासाठी उपयुक्त ठरते.

1.2 पायथन बिल्डिंग ब्लॉक (Python building blocks)

• आयडेंटिफायर (Identifiers):

पायथन आयडेंटिफायर हे एक नाव आहे जे व्हेरिएबल, फंक्शन, वर्ग, मॉड्यूल किंवा इतर ऑब्जेक्ट ओळखण्यासाठी वापरले जाते. आयडेंटिफायरची सुरुवात कॅपिटल लेटर A ते Z किंवा स्मॉल लेटर a ते z किंवा अधोरेखित अक्षराने होते आणि त्यानंतर शून्य किंवा अधिक अक्षरे, अधोरेखित आणि अंक (0 ते 9) असतात.

पायथन आयडेंटिफायर्समध्ये @, \$ आणि % सारख्या चिन्हांना परवानगी देत नाही.

पायथन आयडेंटिफायर्ससाठी नामकरण नियम –

- पायथन क्लासची नावे वरच्या अक्षराने(uppercase) सुरू होतात. इतर सर्व आयडेंटिफायर लोअरकेस अक्षराने (lowercase letter) प्रारंभ करतात.
- एकाच अग्रगण्य अधोरेखनासह आयडेंटिफायर सुरू करणे सूचित करते की आयडेंटिफायर खाजगी आयडेंटिफायर आहे.
- दोन अग्रगण्य अधोरेखकांसह आयडेंटिफायर सुरू करणे एक मजबूत खाजगी आयडेंटिफायर दर्शविते.
- जर आयडेंटिफायर देखील दोन मागच्या अधोरेखनांसह संपत असेल तर आयडेंटिफायर हे भाषा-परिभाषित विशेष नाव आहे.
- **किवर्ड्स (Keywords)**
 - किवर्ड्स हे राखीव(reserve) शब्द आहेत. सर्व पायथन किवर्ड्स केवळ स्मॉल लेटर मध्ये लिहिले जातात असतात.

Table 1.1 : किवर्ड्स (Keywords)

and	as	assert
break	class	continue
def	del	elif
else	except	false

finally	for	from
global	If	import
In	Is	lambda
none	nonlocal	Not
Or	pass	Raise
return	true	Try
while	with	Yield

- **इंडेंटेशन (Indentation):**

पायथन इंडेंटेशन म्हणजे कोडच्या विशिष्ट ब्लॉकमध्ये विधानापूर्वी जागा जोडणे. दुसऱ्या शब्दात, उजवीकडे समान जागा असलेली सर्व विधाने एकाच कोड ब्लॉकची आहेत.

इंडेंटेशन ही पायथनची एक अतिशय महत्वाची संकल्पना आहे कारण पायथन कोड योग्यरित्या इंडेंटेशन केल्याशिवाय आपल्याला इंडेंटेशन एरर दिसेल आणि कोड संकलित होणार नाही.

पायथन इंडेंटेशन हा पायथन दुभाषियाला सांगण्याचा एक मार्ग आहे की विधानांचा गट कोडच्या विशिष्ट ब्लॉकशी संबंधित आहे. ब्लॉक हे या सर्व विधानांचे मिश्रण आहे.

उदाहरण –

```
if true:
    print("true")
    print("answer")

else:
    print("false")
    print("answer")
```

- **व्हेरिएबल्स (variables):**

पायथन व्हेरिएबल्स ही पायथन प्रोग्राममध्ये मूल्ये संग्रहित करण्यासाठी वापरली जाणारी राखीव मेमरी स्थाने आहेत. याचा अर्थ असा की जेव्हा आपण व्हेरिएबल तयार करता तेव्हा आपण मेमरीमध्ये काही जागा राखून ठेवता. व्हेरिएबलच्या डेटा प्रकाराच्या आधारे, पायथन इंटरप्रेटर मेमरी चे वाटप करतो आणि राखीव मेमरीमध्ये काय संग्रहित केले जाऊ शकते हे ठरवते. म्हणूनच, पायथन व्हेरिएबल्सला वेगवेगळे डेटा प्रकार देऊन, आपण या व्हेरिएबल्समध्ये इंटेजर्स(integers), डेसिमलस (decimals) किंवा कॅरेक्टर्स (characters) संग्रहित करू शकता. पायथन व्हेरिएबल्सला मेमरी स्पेस राखीव ठेवण्यासाठी स्पष्ट घोषणेची आवश्यकता नसते किंवा आपण व्हेरिएबल तयार करण्यासाठी म्हणू शकता. जेव्हा आपण त्यास मूल्य देता तेव्हा पायथन व्हेरिएबल आपोआप तयार होते. समान चिन्ह (=) व्हेरिएबलला मूल्ये प्रदान करण्यासाठी वापरले जाते. = ऑपरेटरच्या डाव्या बाजूचे ऑपरेंड हे व्हेरिएबलचे नाव आहे आणि = ऑपरेटरच्या उजव्या बाजूला ऑपरेंड हे व्हेरिएबलमध्ये साठवलेले मूल्य आहे.

उदाहरण:

खालील उदाहरण variables विविध प्रकार (an integer, a float, and a string) तयार करते.

```
counter = 100      # Creates an integer variable
miles = 1000.0     # Creates a floating-point variable
name = "Zara Ali"  # Creates a string variable
```

Once we create a Python variable and assign a value to it, we can print it using print() function.

खालील उदाहरण prints variables विविध प्रकार (an integer, a float, and a string) तयार करते.

```
counter = 100      # Creates an integer variable
miles = 1000.0     # Creates a floating-point variable
name = "Zara Ali"  # Creates a string variable
print(counter)
print(miles)
print(name)
```

Output:

```
100
1000.0
Zara Ali
```

• कमेंट्स (Comments):

पायथनमधील कमेंट्स कोडमधील ओळी आहेत ज्याप्रोग्रामच्या अंमलबजावणीदरम्यान कंपायलर द्वारे दुर्लक्षित केल्या जातात. कमेंट्स कोडची अंडर स्टँडिंग वाढवतात आणि प्रोग्रामरला कोड खूप काळजीपूर्वक समजण्यास मदत करतात.

पायथनमधील कमेंट्स चे प्रकार**1. एक-ओळी कमेंट्स (Single-Line Comments):**

पायथन सिंगल-लाइन कमेंट्स हॅशटॅग चिन्हाने (#) सुरू होते आणि ओळीच्या शेवटापर्यंत टिकते. जर कमेंट्स एका ओळीपेक्षा जास्त असेल तर पुढच्या ओळीवर हॅशटॅग टाका आणि पायथन कमेंट्स चालू ठेवा. व्हेरिबल्स, फंक्शन घोषणा आणि अभिव्यक्तींसाठी संक्षिप्त स्पष्टीकरण प्रदान करण्यासाठी पायथनच्या सिंगल लाईन कमेंट्स उपयुक्त ठरल्या आहेत. सिंगल ओळी कमेंट्स दर्शविणारे खालील कोड स्निपेट पहा:

उदाहरण:

```
# Print "Information Technology" to console
Print ("Information Technology ")
```

2. मल्टीलाइन कमेंट्स (Multi-Line Comments)

पायथन मल्टीलाइन कमेंटसाठी पर्याय प्रदान करत नाही. तथापि, असे वेगवेगळे मार्ग आहेत ज्याद्वारे आपण मल्टीलाइन कमेंट्स लिहू शकतो.

a) एकाधिक हॅशटॅग वापरून मल्टीलाइन कमेंट्स (#)

पायथनमध्ये मल्टीलाइन कमेंट्स लिहिण्यासाठी आम्ही एकाधिक हॅशटॅग (#) करू शकतो. प्रत्येक ओळ ही एक ओळीची कमेंट्स मानली जाईल.

उदाहरण:

```
# Python program to demonstrate
# multiline comments
Print ("Multiline comments")
```

b) स्ट्रिंग शाब्दिक वापरून मल्टीलाइन टिप्पण्या

उदाहरण:

```
""" Python program to demonstrate
multiline comments"""
print ("Multiline comments")
```

1.3 पायथन डेटा प्रकार (python data types)

पायथन डेटा प्रकार व्हेरिएबलचा प्रकार परिभाषित करण्यासाठी वापरले जातात. हे वर्णन करते की आपण व्हेरिएबलमध्ये कोणत्या प्रकारचा डेटा संग्रहित करू. मेमरीमध्ये साठवलेला डेटा अनेक प्रकारचा असू शकतो.

1. नंबर्स (Numbers):

पायथन नंबर डेटा प्रकार संख्यात्मक मूल्ये संग्रहित करतात. जेव्हा आपण त्यांना मूल्य देता तेव्हा संख्या प्रकार तयार केल्या जातात.

उदाहरण:

```
var 1 = 1                # int डेटा प्रकार
var 2 = true             # Boolean डेटा प्रकार
var 3 = 10.023           # float डेटा प्रकार
var 4 = 10+3j            # complex डेटा प्रकार
```

पायथन चार वेगवेगळ्या संख्यात्मक प्रकारांचे समर्थन करते आणि त्यापैकी प्रत्येकाचे पायथन लायब्ररीमध्ये बिल्ट-इन वर्ग आहेत, ज्यांना अनुक्रमे आयएनटी, बूल, फ्लोट आणि कॉम्प्लेक्स म्हणतात.

2. स्ट्रिंग (String):

पायथन स्ट्रिंग एक किंवा अधिक युनिकोड वर्णांचा एक अनुक्रम आहे, जो एकल, दुहेरी किंवा तिहेरी कोटेशन चिन्हांमध्ये बंदिस्त आहे (ज्याला उलटा कोमा देखील म्हणतात). पायथन स्ट्रिंग्स अपरिवर्तनीय असतात ज्याचा अर्थ असा आहे की जेव्हा आपण स्ट्रिंग्सवर ऑपरेशन करता तेव्हा आपण नेहमीच विद्यमान स्ट्रिंगमध्ये बदल करण्याऐवजी त्याच प्रकारची नवीन स्ट्रिंग ऑब्जेक्ट तयार करता. तोपर्यंत सिंगल किंवा डबल किंवा ट्रिपल कोट्सला फरक पडत नाही.

स्ट्रिंग हा एक नॉन-न्यूमेरिक डेटा प्रकार आहे. साहजिकच त्यावर आपण अंकगणिताची कामे करू शकत नाही. तथापि, स्लाइसिंग आणि कॉन्कॅटेशन सारख्या क्रिया केल्या जाऊ शकतात. पायथनचा str वर्ग(class) स्ट्रिंग प्रोसेसिंगसाठी अनेक उपयुक्त पद्धती परिभाषित करतो. प्लस (+) चिन्ह स्ट्रिंग कॉन्कॅटेशन ऑपरेटर आहे आणि तारांकन (*) पायथनमधील पुनरावृत्ती ऑपरेटर आहे.

उदाहरण:

```
str = 'Hello World!'
print (str)                # Prints complete string
```

```

print(str[0])           # Prints first character of the string
print(str[2:5])         # Prints characters starting from 3rd to 5th
print(str[2:])          # Prints string starting from 3rd character
print(str * 2)          # Prints string two times
print(str + "TEST")     # Prints concatenated string

```

Output:

```

Hello World!
Hallo World!
Hello World!Hello World!
Hello World!TEST

```

3. टपल्स (Tuples):

टपलचा वापर एकाच व्हेरिएबलमध्ये अनेक वस्तू साठविण्यासाठी केला जातो. टपल हा एक संग्रह आहे जो क्रमबद्ध आणि अपरिवर्तनीय आहे. टपल राऊंड ब्रॅकेट () लिहिलेले असतात.

पायथन टपलमध्ये कोमाद्वारे विभक्त केलेली अनेक मूल्ये असतात. टपल राऊंड ब्रॅकेट मध्ये लिहिले जातात (...).

टपल हादेखील एक क्रम आहे, म्हणून टपलमधील प्रत्येक वस्तूला संग्रहातील त्याच्या स्थानाचा संदर्भ देणारा निर्देशांक असतो. निर्देशांक पासून सुरू होतो.

उदाहरण:

```

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')
print (tuple)           # Prints the complete tuple
print (tuple[0])        # Prints first element of the tuple
print (tuple[1:3])      # Prints elements of the tuple starting from 2nd till 3rd
print (tuple[2:])       # Prints elements of the tuple starting from 3rd element
print (tinytuple * 2)    # Prints the contents of the tuple twice
print (tuple + tinytuple) # Prints concatenated tuples

```

Output:

```

('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')

```

4. लिस्ट (List):

लिस्ट सर्वात अष्टपैलू कंपाऊंड डेटा प्रकार आहेत. लिस्टमध्ये कॉमाद्वारे विभक्त केलेल्या आणि चौकोनी ब्रॅकेट च्या मध्ये असतात ([]). पायथन लिस्टशी संबंधित सर्व वस्तू वेगवेगळ्या डेटा प्रकारच्या असू शकतात. एका लिस्टमध्ये साधे आकडे, धागे, टपल, शब्दकोश अशा वस्तू असू शकतात. पायथन लिस्टमध्ये

साठवलेली मूल्ये स्लाइस ऑपरेटर ([] आणि [:]) वापरून एक्सेस केली जाऊ शकतात ज्यात अनुक्रमणिका सूचीच्या सुरुवातीला 0 पासून सुरू होतात आणि -1 च्या शेवटी कार्य करतात.

उदाहरण:

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylis = [123, 'john']
print (list)           # Prints complete list
print (list[0])         # Prints first element of the list
print (list[1:3])       # Prints elements starting from 2nd till 3rd
print (list[2:])        # Prints elements starting from 3rd element
print (tinylis * 2)      # Prints list two times
print (list + tinylis)   # Prints concatenated lists
```

Output:

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

5. डिक्शनरी (Dictionary):

डिक्शनरीचा उपयोग कीमध्ये डेटा मूल्ये संग्रहित करण्यासाठी केला जातो key: value मूल्य जोड्या. डिक्शनरी हा एक संग्रह आहे जो ऑर्डर केलेला*, बदलण्यायोग्य आहे आणि डुप्लिकेटला परवानगी देत नाही. जोड्या कॉमाद्वारे विभक्त केल्या जातात आणि करली ब्रॅकेट च्या आत ठेवल्या जातात {}. की आणि मूल्य यांच्यात मॅपिंग स्थापित करण्यासाठी, सेमीकोलन ':' चिन्ह दोघांमध्ये ठेवले जाते.

डिक्शनरी करली ब्रॅकेट ({}) ने वेढलेले असतात आणि चौकोनी ब्रॅकेट ([]) वापरून मूल्ये निश्चित केली जाऊ शकतात आणि प्रवेश केला जाऊ शकतो. डिक्शनरी बदलण्यायोग्य आहेत, म्हणजे डिक्शनरी तयार झाल्यानंतर आपण वस्तू बदलू शकतो, जोडू शकतो किंवा काढून टाकू शकतो.

उदाहरण:

```
dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
print (dict['one'])     # Prints value for 'one' key
print (dict[2])        # Prints value for 2 key
print (tinydict)       # Prints complete dictionary
print (tinydict.keys()) # Prints all the keys
print (tinydict.values()) # Prints all the values
```

Output:

```
This is one
This is two
```

```
{'name': 'john', 'code': 6734, 'dept': 'sales'}
dict_keys(['name', 'code', 'dept'])
dict_values(['john', 6734, 'sales'])
```

1.4 बेसिक ऑपरेटर्स (Basic Operators):

1. अरीथमेटिक ऑपरेटर्स (Arithmetic Operators):

अरीथमेटिक ऑपरेटर हे बायनरी ऑपरेटर आहेत या अर्थाने ते दोन ऑपरेंडवर कार्य करतात. पायथन पूर्णपणे मिश्र अरीथमेटिक समर्थन करतो. आयएनटी आणि फ्लोट च्या अरीथमेटिक ऑपरेशनचा परिणाम म्हणजे फ्लोट. फ्लोट आणि कॉम्प्लेक्सचा परिणाम हा एक गुंतागुंतीचा आकडा आहे.

पायथनमध्ये उपलब्ध असलेल्या सर्व अरीथमेटिक ऑपरेटर्सची यादी देणारा तक्ता खालीलप्रमाणे आहे:

Table 1.2 : अरीथमेटिक ऑपरेटर (Arithmetic Operators)

ऑपरेटर (Operator)	नाम (Name)	उदाहरण (Example)
+	बेरीज (Addition)	$a + b = 30$
-	वजाबाकी (Subtraction)	$a - b = -10$
*	गुणाकार (Multiplication)	$a * b = 200$
/	भागाकार (Division)	$b / a = 2$
%	मोड्यूलस (Modulus)	$b \% a = 0$
**	घातांक (Exponent)	$a ** b = 10 ** 20$
//	फ्लोर डिव्हिजन (Floor Division)	$9 // 2 = 4$

2. रिलेशनल /कंपेरिजन ऑपरेटर्स (Relational/Comparison Operators):

दोन मूल्यांची तुलना करण्यासाठी कंपेरिजन ऑपरेटर वापरले जातात. कंपेरिजन ऑपरेटर्सना रिलेशनल ऑपरेटर्स देखील म्हणतात. पायथनमध्ये उपलब्ध असलेल्या सर्व कंपेरिजन ऑपरेटर्सची यादी देणारी सारणी खालीलप्रमाणे आहे:

Table 1.3: . रिलेशनल /कंपेरिजन ऑपरेटर(Relational/Comparison Operators)

ऑपरेटर (Operator)	नाम (Name)	उदाहरण (Example)
<	पेक्षा कमी (Less than)	$a < b$
>	पेक्षा जास्त (Greater than)	$a > b$
<=	पेक्षा कमी किंवा समतुल्य (Less than or equal to)	$a <= b$
>=	पेक्षा जास्त किंवा समतुल्य (Greater than or equal to)	$a >= b$
= =	च्या बरोबरीने आहे (Is equal to)	$a == b$
!=	च्या बरोबरीने नाही (Is not equal to)	$a != b$

3. असाइनमेंट ऑपरेटर्स (Assignment Operators):

व्हेरिएबल्सला मूल्ये प्रदान करण्यासाठी असाइनमेंट ऑपरेटर्सचा वापर केला जातो.

Table 1.4: असाइनमेंट ऑपरेटर्स (Assignment Operators)

ऑपरेटर (Operator)	उदाहरण (Example)	जसे की (Same As)
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3
:=	Print (x := 3)	x = 3 print(x)

4. लॉजिकल ऑपरेटर्स (Logical Operators):

लॉजिकल ऑपरेटरचा वापर सशर्त विधाने एकत्रित करण्यासाठी केला जातो:

Table 1.5: लॉजिकल ऑपरेटर्स (Logical Operators)

ऑपरेटर (Operator)	वर्णन (Description)	उदाहरण (Example)
अँड (AND)	अँड मधील दोन्ही विधाने True असतील तर रिटर्न true (Returns True if both statements are true)	x < 5 and x < 10
ऑर (OR)	ऑर मधील जर एखादे विधान TRUE असेल तर रिटर्न TRUE ठरतो (Returns True if one of the statements is true)	x < 5 or x < 4
नॉट (NOT)	नॉट मधील विधान false असेल तर रिटर्न TRUE, विधान true असेल तर रिटर्न false. (Reverse the result, returns False if the result is true)	NOT (x < 5 and x < 10)

5. बिटवाइज ऑपरेटर्स (Bitwise Operators):

बिटवाइज ऑपरेटर्सचा वापर (बायनरी) संख्यांची तुलना करण्यासाठी केला जातो.

Table 1.6 : बिटवाइज ऑपरेटर्स (Bitwise Operators)

ऑपरेटर (Operator)	नाम (Name)	वर्णन (Description)	उदाहरण (Example)
&	AND	दोन्ही बिट्स 1 असल्यास प्रत्येक बिट 1 वर सेट करा (Sets each bit to 1 if both bits are 1)	$x \& y$
	OR	जर दोन बिट्सपैकी एक 1 असेल तर प्रत्येक बिट 1 वर सेट करा (Sets each bit to 1 if one of two bits is 1)	$x y$
^	XOR	जर दोन बिट्सपैकी फक्त एक 1 असेल तर प्रत्येक बिट 1 वर सेट करा (Sets each bit to 1 if only one of two bits is 1)	$x \wedge y$
~	NOT	सर्व बिट्स कॉम्पलीमेंटरी करतात (Inverts all the bits)	$\sim x$
<<	Zero fill left shift	उजवीकडून शून्यांना आत ढकलून डावीकडे वळा आणि डावे तुकडे खाली पडू द्या (Shift left by pushing zeros in from the right and let the leftmost bits fall off)	$x \ll 2$
>>	Signed right shift	डावीकडून डाव्या बिटच्या प्रती आत ढकलून उजवीकडे सरका आणि उजवे तुकडे खाली पडू द्या (Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off)	$x \gg 2$

6. मेम्बरशिप ऑपरेटर्स (Membership Operators):

एखाद्या ऑब्जेक्टमध्ये अनुक्रम सादर केला आहे की नाही हे तपासण्यासाठी मेम्बरशिप ऑपरेटरचा वापर केला जातो.

Table 1.7: सदस्यता ऑपरेटर्स (Membership Operators)

ऑपरेटर (Operator)	वर्णन (Description)	उदाहरण (Example)
in	जर ऑब्जेक्टमध्ये स्पेसिफाइड value असेल तर रिटर्न true ठरतो (Returns True if a sequence with the specified value is present in the object)	$x \text{ in } y$
not in	जर ऑब्जेक्टमध्ये निर्दिष्ट value असलेला नसल्यास रिटर्न true ठरतो (Returns True if a sequence with the specified value is not present in the object)	$x \text{ not in } y$

6. आयडेंटिटी ऑपरेटर्स (Identity Operators):

आयडेंटिटी ऑपरेटर्सचा वापर ऑब्जेक्ट्सची तुलना करण्यासाठी केला जातो, जर ते समान असतील तर नाही, परंतु जर ते खरोखर समान ऑब्जेक्ट असतील, समान मेमरी स्थानासह.

ऑपरेटर (Operator)	वर्णन (Description)	उदाहरण (Example)
is	दोन्ही व्हेरिएबल्स एकच ऑब्जेक्ट असतील तर रिटर्न true (Returns True if both variables are the same object)	x is y
is not	दोन्ही व्हेरिएबल्स एकच ऑब्जेक्ट नसतील तर रिटर्न true (Returns True if both variables are not the same object)	x is not y

7. ऑपरेटर अग्रक्रम (Operator Precedence):

ऑपरेटर अग्रक्रम त्या क्रमाची व्याख्या करतो ज्यामध्ये ऑपरेटर्सचे मूल्यांकन केले जाते. पायथनमधील केवळ अंकगणित ऑपरेटर्सचा विचार केल्यास, पारंपारिक बीओडीएमएस नियम पायथन इंटरप्रेटरद्वारे देखील वापरला जातो, जिथे ब्रॅकेट चे मूल्यांकन प्रथम केले जाते, विभाजन आणि गुणाकार ऑपरेटर नंतर जोडले जातात आणि त्यानंतर जोड आणि वजाबाकी ऑपरेटर असतात.

अग्रक्रम क्रम खालील तक्त्यात वर्णन केला आहे, ज्याची सुरुवात शीर्षस्थानी सर्वोच्च अग्रक्रमाने होते.

Table 1.8: ऑपरेटर अग्रक्रम (Operator Precedence)

ऑपरेटर (Operator)	वर्णन (Description)
()	ब्रॅकेट (Parentheses)
**	एक्सपोनेन्टिएशन (Exponentiation)
+x, -x, ~x	उनरी प्लस, उनरी माइनस आणि बिटवाइज नॉट (Unary plus, unary minus, and bitwise NOT)
* / // %	गुणाकार, डीविजन, फ्लोर डीविजन आणि मॉड्युल्स (Multiplication, division, floor division, and modulus)
+ -	बेरीज आणि वजाबाकी (Addition and subtraction)
<< >>	बिटवाइज लेफ्ट अँड राइट शिफ्ट (Bitwise left and right shifts)
&	बिटवाइज अँड (Bitwise AND)
^	बिटवाइज एक्सओआर (Bitwise XOR)
	बिटवाइज किंवा (Bitwise OR)
== != > >= < <= is is not in not in NOT	तुलना, ओळख आणि सदस्यत्व ऑपरेटर (Comparisons, identity, and membership operators) लॉजिकल नॉट (Logical NOT)
AND	अँड (AND)
OR	ऑर (OR)

जर दोन ऑपरेटर्सना समान प्राधान्य असेल तर अभिव्यक्तीचे मूल्यांकन डावीकडून उजवीकडे केले जाते.

1.5 कंट्रोल फ्लो (Control flow):

1.5.1 कंडिशनल स्टेटमेंट (Conditional statements):

a. इफ (if) - जर विधान हे सर्वात सोपे निर्णय घेणारे विधान आहे. एखाद्या विशिष्ट विधानाची किंवा विधानांच्या ब्लॉकची अंमलबजावणी होईल की नाही हे ठरविण्यासाठी याचा वापर केला जातो.

मूल्यमापनानंतरची अट खरी किंवा खोटी असेल. जर विधान बुलियन मूल्ये स्वीकारत असेल - जर मूल्य खरे असेल तर ते त्याखालील विधानांचा ब्लॉक अंमलात आणेल अन्यथा नाही.

वाक्यरचना (syntax):

if condition:

statement1

statement2

येथे जर **कंडिशन** true असेल तर ब्लॉक फक्त स्टेटमेंट 1 आपल्या ब्लॉकच्या आत आहे असे मानेल.

उदाहरण:

a = 33

b = 200

if (b > a):

print ("b is greater than a")

Output:

b is greater than a

b. इफ एल्स (IF-ELSE):

जर केवळ विधान आपल्याला सांगते की जर एखादी अट खरी असेल तर ती विधानांचा ब्लॉक अंमलात आणेल आणि जर ती अट चुकीची असेल तर ती करणार नाही. परंतु जर अट खोटी असेल तर आपल्याला दुसरे काही करायचे असेल तर पायथन जर अट खोटी असेल तर आपण कोडचा ब्लॉक अंमलात आणण्यासाठी पायथन या विधानासह else विधान वापरू शकतो. else कीवर्ड अशी कोणतीही गोष्ट पकडतो जी आधीच्या अटीद्वारे पकडली जात नाही.

वाक्यरचना (syntax):

if (condition):

Executes this block if condition is true

else:

Executes this block if condition is false

Example:

i = 20

if (i < 15):

print("i is smaller than 15")

print("i'm in if Block")

else:

print("i is greater than 15")

print("i'm in else Block")

print("i'm not in if and not in else Block")

Output:

i is greater than 15

i'm in else Block

i'm not in if and not in else Block

c. नेस्टेड इफ (Nested if):

जर विधाने असतील तर याचा अर्थ असा आहे की आपण सशर्त वापरू शकतो अन्यथा जर अस्तित्वात असलेल्या विधानाच्या आत विधान असेल तर. अशी परिस्थिती उद्भवू शकते जेव्हा एखादी स्थिती सत्यात उतरल्यानंतर आपण दुसऱ्या स्थितीची तपासणी करू इच्छिता. अशा तऱ्हेने तुम्ही नेस्टेड चा वापर करू शकता. नेस्टेड मध्ये जर केले तर आपणास एक एल्स इफ नाहीतर दुसर् याच्या आत कंस्ट्रक्ट करा.

वाक्यरचना (Syntax):

```
if expression1:
```

```
    statement(s)
```

```
    if expression2:
```

```
        statement(s)
```

```
    elif expression3:
```

```
        statement(s)3
```

```
    else
```

```
        statement(s)
```

```
elif expression4:
```

```
    statement(s)
```

```
else:
```

```
    statement(s)
```

Example:

```
num=8
```

```
print ("num = ", num)
```

```
if num%2==0:
```

```
    if num%3==0:
```

```
        print ("Divisible by 3 and 2")
```

```
    else:
```

```
        print ("divisible by 2 not divisible by 3")
```

```
else:
```

```
    if num%3==0:
```

```
        print ("divisible by 3 not divisible by 2")
```

```
    else:
```

```
        print ("not Divisible by 2 not divisible by 3")
```

Output:

```
num = 8
```

```
divisible by 2 not divisible by 3
```

1.5.2 लूपिंग इन पायथन (Looping in python):

a. व्हाईल लूप (while loop):

पायथन प्रोग्रामिंग भाषेत काही काळ लूप स्टेटमेंट वारंवार लक्ष्य विधान अंमलात आणते जोपर्यंत दिलेली बुलियन कंडिशन खरी आहे.

वाक्यरचना(syntax):

while expression:

statement(s)

कीवर्डनंतर बुलियन कंडिशन आणि नंतर कोलन चिन्हाद्वारे विधानांचा एक इनडेंटेड ब्लॉक सुरू केला जातो. येथे, विधान (विधाने) एकच विधान असू शकतात किंवा एकसमान इंडेंट असलेल्या विधानांचा ब्लॉक असू शकतो. अट कोणतीही कंडिशन असू शकते आणि सत्य म्हणजे शून्य नसलेले मूल्य. बुलियन कंडिशन खरी असताना लूप इटेट होतो. कंडिशन खोटी होताच, प्रोग्राम नियंत्रण लूपनंतर लगेचच रेषेवर जाते.

उदाहरण:

```
#Print i as long as i is less than 6:
```

```
i = 1
```

```
while i < 6:
```

```
    print(i)
```

```
    i += 1
```

Output:

```
1
2
3
4
5
```

b. फॉर लूप (for loop): फॉर लूप चा उपयोग अनुक्रमावर (म्हणजे एकतर लिस्ट, टपल, डिक्शनरी किंवा स्ट्रिंग) करण्यासाठी केला जातो.

वाक्यरचना(Syntax):

for iterating variable in sequence:

statements(s)

जर एखाद्या अनुक्रमामध्ये कंडिशन लिस्ट असेल तर प्रथम त्याचे मूल्यांकन केले जाते. त्यानंतर, अनुक्रमातील पहिली आयटम (0 व्या निर्देशांकावर) इटरेटिंग व्हेरिएबल `iterating_var` दिली जाते.

पुढे, स्टेटमेंट ब्लॉक अंमलात आणला जातो. यादीतील प्रत्येक आयटम `iterating_var` देण्यात आली आहे आणि संपूर्ण अनुक्रम संपेपर्यंत स्टेटमेंट ब्लॉक अंमलात आणला जातो.

उदाहरण :

```
#Print each fruit in a fruit list:
```

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in fruits:
```

```
    print(x)
```

Output:

```
apple
banana
cherry
```

c. नेस्टेड लूप (Nested loop):

जेव्हा आपण लूप स्टेटमेंटमध्ये एक किंवा अधिक लूप लिहिता ज्यास नेस्टेड लूप म्हणून ओळखले जाते. मुख्य लूपला बाह्य लूप मानले जाते आणि बाह्य लूपच्या आतील लूपला आतील लूप म्हणून ओळखले जाते. पायथन प्रोग्रामिंग भाषा दुसऱ्या लूपच्या आत एक लूप वापरण्याची परवानगी देते

• नेस्टेड फॉर लूप (Nested for loop):

लूपसाठी एक किंवा अधिक आतील असलेल्या लूपसाठी लूपसाठी नेस्टेड केले जाते.

वाक्यरचना(syntax):

```
for iterating_var in sequence:
    for iterating_var in sequence:
        statements(s)
    statements(s)
```

नेस्टेड व्हाईल लूप (Nested while loop):

लूपमध्ये एक किंवा अधिक आतील लूप असतात तर लूप नेस्टेड असतात.

वाक्यरचना(Syntax):

```
while expression:
    while expression:
        statement(s)
    statement(s)
```

लूप मॅनिप्युलेशन स्टेटमेंट (Loop manipulation statement):

- **कंटिन्यू स्टेटमेंट (Continue statement):** प्रोग्राम ब्लॉकची अंमलबजावणी वगळण्यासाठी कंटिन्यूअर स्टेटमेंटचा वापर केला जातो आणि पुढील पुनरावृत्ती सुरू करण्यासाठी सध्याच्या लूपच्या सुरूवातीस नियंत्रण परत केले जाते. जेव्हा सामोरे जावे लागते, तेव्हा लूप सध्याच्या पुनरावृत्तीतील उर्वरित विधानांची अंमलबजावणी न करता पुढील पुनरावृत्ती सुरू करते. कंटिन्यूअर स्टेटमेंट चा वापर करताना आणि लूपसाठी दोन्ही प्रकारे केला जाऊ शकतो.

वाक्यरचना(Syntax):

```
continue
```

उदाहरण:

```
for letter in 'Python':
    if letter == 'h':
        continue
    print ('Current Letter:', letter)
```

Output:

```
Current Letter: P
```

```
Current Letter: y
```

Current Letter: t

Current Letter: o

Current Letter: n

- **पास स्टेटमेंट (pass statement):**

जेव्हा एखाद्या विधानाची आवश्यकता असते तेव्हा पास स्टेटमेंट वापरले जाते परंतु आपल्याला कोणतीही कमांड किंवा कोड अंमलात आणायचा नसतो. हे एक शून्य ऑपरेशन आहे; ती अंमलात आल्यावर काहीच होत नाही. पायथन पास स्टेटमेंट अशा ठिकाणी देखील उपयुक्त आहे जिथे आपला कोड शेवटी जाईल, परंतु अद्याप लिहिला गेला नाही.

वाक्यरचना(syntax):

```
pass
```

उदाहरण:

```
for letter in 'Python':
    if letter == 'h':
        pass
    print ('This is pass block')
    print ('Current Letter:', letter)
print ("Good bye!")
```

Output:

Current Letter: P

Current Letter: y

Current Letter: t

This is pass block

Current Letter: h

Current Letter: o

Current Letter: n

Good bye!

- **ब्रेक स्टेटमेंट (break statement):**

ब्रेक स्टेटमेंटचा वापर करंट लूप समाप्त करण्यासाठी केला जातो आणि C प्रोग्रामिंग मधील पारंपारिक ब्रेक स्टेटमेंटप्रमाणेच पुढील स्टेटमेंटवर अंमलबजावणी पुन्हा सुरू केली जाते.

पायथन ब्रेक स्टेटमेंटचा सर्वात सामान्य वापर म्हणजे जेव्हा एखादी बाह्य स्थिती उद्भवते तेव्हा लूपमधून बाहेर पडण्याची आवश्यकता असते. ब्रेक स्टेटमेंट पायथनमध्ये लूपसाठी वापरले जाऊ शकते.

वाक्यरचना(syntax):

```
break
```

उदाहरण:

```
for letter in 'Python':
    if letter == 'h':
        break
    print ('Current Letter:', letter)
```

Output:

Current Letter: P

Current Letter: y

Current Letter: t

संदर्भ सूची (References) :

1. <https://pynative.com/python-control-flow-statements/>
2. <https://www.freecodecamp.org/news/control-flow-in-programming-b9fb4f4539c/>
3. <https://builtin.com/software-engineering-perspectives/control-flow>
4. https://www.tutorialspoint.com/python/python_control_flow.html
5. <https://docs.python.org/3/tutorial/controlflow.html>
6. https://www.w3schools.com/python/python_conditions.asp
7. <https://www.programiz.com/python-programming>

युनिट -2

पायथनमधील स्पेसिफिक डेटा स्ट्रक्चर्स अँड फंक्शन्स

(Python Specific Data Structure and Functions)

विषय निष्पत्ती (Course Outcome): पायथनमधील अनुक्रम संरचनांवर ऑपरेशन करा.

घटक निष्पत्ती (Theory Learning Outcome):

1. दिलेल्या समस्येसाठी लिस्ट हाताळण्यासाठी पायथन प्रोग्राम विकसित करा.
2. दिलेल्या प्रॉब्लेमसाठी टपल्स हाताळण्यासाठी पायथन प्रोग्राम विकसित करा.
3. दिलेल्या प्रॉब्लेम मध्ये सेट हाताळण्यासाठी पायथन प्रोग्राम लिहा.
4. दिलेल्या प्रॉब्लेमसाठी डिक्शनरी हाताळण्यासाठी पायथन प्रोग्राम लिहा.
5. दिलेल्या समस्येसाठी संबंधित युझर डिफाइन फंक्शन विकसित करा.

2.1 लिस्ट (Lists)

- पायथन मध्ये सूची ही इतर भाषांमध्ये घोषित केलेल्या डायनॅमिकली आकाराच्या अरेसारख्या असतात (C++ मध्ये वेक्टर आणि Java मध्ये Array List).
- सोप्या भाषेत, सूची म्हणजे [] मध्ये बंद केलेल्या आणि स्वल्पविरामाने विभक्त केलेल्या गोष्टींचा संग्रह. सूची एक अनुक्रम डेटा प्रकार आहे जो डेटा संग्रहित करण्यासाठी वापरला जातो.
- सूची आयटम ऑर्डर केलेले असतात, बदलण्यायोग्य असतात आणि डुप्लिकेट मूल्यांना अनुमती देतात.

2.1.1 लिस्ट डिफाइन करणे (Defining List)

लिस्ट खालीलप्रमाणे परिभाषित केली जाऊ शकते.

```
L1 = ["MSBTE", 102, "SANGAMNER"]
```

```
L2 = [1, 2, 3, 4, 5, 6]
```

```
L3 = [1, "JAVA"]
```

```
Var = ["Hello", "to", "All"]
```

```
print(Var)
```

आउटपुट (Output): ["Hello", "to", "All"]

लिस्ट सर्वात सोप्या कंटेनर आहेत जे पायथन भाषेचा अविभाज्य भाग आहेत. सूची नेहमी एकसमान नसतात ज्यामुळे ते पायथनमधील सर्वात महत्वाचे साधन बनते. एका लिस्टमध्ये इंटेजर्स, स्ट्रिंग्स, तसेच ऑब्जेक्ट्स सारखे डेटा प्रकार असू शकतात. लिस्ट बदलण्यायोग्य असतात, आणि म्हणूनच, त्यांच्या निर्मितीनंतरही त्या बदलल्या जाऊ शकतात.

2.1.2 लिस्ट मधील व्हॅल्यू ऍक्सेस करणे (Accessing values in list)

पायथन लिस्टमध्ये व्हॅल्यू ऍक्सेस करण्यासाठी, आपण लिस्टच्या इंडेक्सचा (index) वापर करू शकतो.

खाली याचे उदाहरण दिले आहे

लिस्टमध्ये पहिली व्हॅल्यू झिरो (0) इंडेक्सला दाखविते होते.

उदाहरण(eg.):

लिस्टचा दुसरा आयटम मुद्रित करा:


```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

आउटपुट(Output):

banana

वरील उदाहरणामध्ये पहिल्या एलिमेंट चा इंडेक्स 0 आहे.

- **नेगेटिव्ह इंडेक्सिंग (Negative Indexing)** : नेगेटिव्ह इंडेक्सिंग म्हणजेच शेवटीपासून सुरुवात करणे . -1 हा इंडेक्स शेवटच्या एलिमेंट संदर्भित करतो, -2 हा इंडेक्स दुसरी शेवटच्या वस्तूसाठी संदर्भित करतो इत्यादी. उदाहरण(eg.): सूचीची शेवटची वस्तू प्रिंट करा:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

- **इंडेक्सची रेंज (Range of Indexes)**: तुम्ही श्रेणी सूचित करून सुरुवात करण्याचे आणि शेवट करण्याचे ठरवू शकता . रेंज माहिती असेल तर, त्यामध्ये दिलेल्या सुरुवातीच्या इंडेक्स पासून ते शेवटी असणाऱ्या इंडेक्स पर्यंतचे एलिमेंट मिळू शकतात.

उदाहरण(eg.): तिसऱ्या , चौथ्या आणि पाचव्या एलिमेंट दाखवा :

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

आउटपुट(Output):

```
['cherry', 'orange', 'kiwi']
```

उदाहरण(eg.): या उदाहरणात, "kiwi" समाविष्ट करून प्रारंभापासून वस्तूंना परत करतो:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

आउटपुट(Output):

```
['apple', 'banana', 'cherry', 'orange']
```

उदाहरण(eg.): या उदाहरणात, "cherry" पासून सुरू होऊन समाप्तीपर्यंतच्या वस्तूंना परत करतो:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

आउटपुट(Output):

```
['cherry', 'orange', 'kiwi', 'melon', 'mango']
```

उदाहरण (eg.): या उदाहरणात, "orange" (-4) पासून सुरू होऊन "mango" (-1) समाप्तीपर्यंतच्या वस्तूंना परत करतो:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

आउटपुट (Output):-

```
['orange', 'kiwi', 'melon']
```

- **लिस्टमधील व्हॅल्यू डिलीट करणे (Deleting values in list):**

लिस्टमध्ये व्हॅल्यू डिलीट करण्यासाठी del कीवर्डचा वापर केला जातो. पायथनने आपल्याला remove() मेथड दिलेली आहे. आपल्याला कोणते व्हॅल्यू लिस्टमधून हटवायचे असल्यास त्या व्हॅल्यूचा इंडेक्स द्यावा लागतो.

- **लिस्टमधील व्हॅल्यू डिलीट करण्याची प्रक्रिया खालील प्रमाणे**

```
List = [0, 1, 2, 3, 4]
```

```
print(List)
```

आउटपुट(Output):

```
[0, 1, 2, 3, 4]
```

```
del List[0]
```

```
print(List)
```

आउटपुट(Output):

```
[1, 2, 3, 4]
```

```
del List[3]
```

```
print(List)
```

आउटपुट(Output):

```
[1, 2, 3]
```

येथे, पहिल्या प्रिंट स्टमेन्ट ने सूचीचे मुख्य प्रदर्शित केले आहे. नंतर del कीवर्डचा वापर करून पहिल्या मुख्य हटवून दुसरी लिस्ट प्रदर्शित केली आहे. आणि शेवटी तिसरे मुख्य हटवून आपल्याला आपल्या निर्देशकानुसार सूची प्रदर्शित केली आहे.

- **लिस्ट अपडेट करणे (Updating lists):** आधी तयार केलेल्या लिस्टमध्ये आपण बदल करू शकतो. त्याला लिस्ट अपडेशन म्हणतात. यासाठी आपल्याला इंडेक्स नंबर द्यावा लागतो आणि त्याची जी व्हॅल्यू आहे ती द्यावी लागते. खाली याचे उदाहरण दिले आहे.

```
List = [1, 2, 3, 4, 5, 6]
```

```
print(List)
```

आउटपुट(Output): [1, 2, 3, 4, 5, 6]

```
List[2] = 10
```

```
print(List)
```

आउटपुट(Output): [1, 2, 10, 4, 5, 6]

```
List[1:3] = [89, 78]
```

```
print(List)
```

आउटपुट(Output): [1, 89, 78, 4, 5, 6]

येथे, पहिल्या प्रिंट स्टमेन्टने सूचीचे मुख्य प्रदर्शित केले आहे. नंतर, तुम्ही एक मुख्यचे अद्यतन सूचीत करण्यासाठी त्याचे स्थानिक इंडेक्स दिले आहे. आणि शेवटी तुम्ही दिलेल्या श्रेणीचे मूल्य सूचीमध्ये अद्यावत केले आहे.

- **बेसिक लिस्ट ऑपरेशन (Basic List Operations)**

पायथनमधील सूचीवरील मूलभूत क्रियांची माहिती घेऊयात.

1. **काँकॅटीनेशन (concatenation):** + ऑपरेटरचा वापर करून लिस्टचे संयोजन करा.

```
# Basic list operations
list1 = [1, 2, 3]
list2 = [4, 5, 6]
# Concatenation
combined_list = list1 + list2
Output
[1, 2, 3, 4, 5, 6]
```

2. **रिपीटेशन (Repetition):** * ऑपरेटरचा वापर करून लिस्टची पुनरावृत्ती करा.

```
# Repetition
repeated_list = list1 * 2
Output
[1, 2, 3, 1, 2, 3]
```

3. **मेम्बरशिप (Membership):** in ऑपरेटरचा वापर करून एखादे आयटम लिस्टमध्ये आहे की नाही ते तपासा.

```
# Membership
is_present = 3 in list1 # True
```

4. **लेंथ (length) :** len() फंक्शनचा वापर करून लिस्टची लांबी मिळवा.

```
# Length
length = len(list1) # 3
```

- **बिल्ट-इन लिस्ट फंक्शन (Built-in List Functions)**

पायथनमध्ये लिस्टसाठी काही अंगभूत फंक्शन्स आहेत.

1. append(): लिस्टमधील एक घटक संघटित करणे.
2. clear(): लिस्टमधील सर्व घटक काढणे.
3. copy(): लिस्टमधील प्रतिलिपी प्राप्त करणे.
4. count(): विशिष्ट घटक संख्या मिळवणे .
5. extend(): एक लिस्टमधील घटक सूचीच्या शेवटी जोडणे.
6. index(): निश्चित घटक पहिल्या घटकाचे इंडेक्स मिळवणे.
7. insert(): निश्चित स्थानावर एक घटक समाविष्ट करणे
8. pop(): निश्चित स्थानावरील घटक काढणे.
9. remove(): विशिष्ट घटकाचे पहिले घटक सूचीतून काढणे.
10. reverse(): लिस्टच्या घटकाची उलट क्रमवारी लावणे
11. sort(): लिस्टमधील घटक क्रमांकानुसार क्रमबद्ध करणे

अशा प्रकारे, आपल्या प्रोग्राम मध्ये सूचीसाठी आवश्यक असणारे फंक्शन्स आपण वापरू शकतो.

1. **अपेंड append():** या मेथडचा उपयोग करून तुम्ही सूचीत एक घटक संघटित करू शकता.

```
my_list = [1, 2, 3]
my_list.append(4)
```

```
print(my_list)
```

आउटपुट(Output): [1, 2, 3, 4]

2. **क्लीअर clear():** या मेथडचा उपयोग करून तुम्ही लिस्टमधील सर्व घटक काढू शकता.

```
my_list = [1, 2, 3]
```

```
my_list.clear()
```

```
print(my_list)
```

आउटपुट(Output): []

3. **कॉपी copy():** या मेथडचा उपयोग करून तुम्ही लिस्टची प्रतिलिपी प्राप्त करू शकता.

```
original_list = [1, 2, 3]
```

```
copied_list = original_list.copy()
```

```
print(copied_list)
```

आउटपुट(Output): [1, 2, 3]

4. **काउंट count():** या मेथडचा उपयोग करून तुम्ही विशिष्ट मूल्याची संख्या मिळवू शकता

```
my_list = [1, 2, 2, 3, 4, 2]
```

```
count_of_2 = my_list.count(2)
```

```
print(count_of_2)
```

आउटपुट(Output): 3

5. **एक्सटेंड extend():** या मेथडचा उपयोग करून तुम्ही एक घटक लिस्टच्या शेवटी जोडू शकता.

```
list1 = [1, 2, 3]
```

```
list2 = [4, 5, 6]
```

```
list1.extend(list2)
```

```
print(list1)
```

आउटपुट(Output): [1, 2, 3, 4, 5, 6]

6. **इंडेक्स index():** या मेथडचा उपयोग करून तुम्ही निश्चित मूल्याचे पहिले इंडेक्स मिळवू शकता.

```
my_list = [10, 20, 30, 20, 40]
```

```
index_of_20 = my_list.index(20)
```

```
print(index_of_20)
```

आउटपुट(Output): 1

7. **इन्सर्ट insert():** या मेथडचा उपयोग करून तुम्ही एक निश्चित स्थानावर एक घटक सामाविष्ट करू शकता.

```
my_list = [1, 2, 3]
```

```
my_list.insert(1, 5)
```

```
print(my_list)
```

आउटपुट(Output): [1, 5, 2, 3]

8. **पॉप pop():** या मेथडचा उपयोग करून तुम्ही निश्चित स्थानावर घटक काढू शकता.

```
my_list = [1, 2, 3, 4]
```

```
popped_value = my_list.pop(2)
print(popped_value)
आउटपुट(Output): 3
print(my_list)
```

आउटपुट(Output): [1, 2, 4]

9. रिमूव remove(): या मेथडचा उपयोग करून तुम्ही विशिष्ट घटकाचे पहिले घटक लिस्टमधून काढू शकता.

```
my_list = [1, 2, 3, 4]
my_list.remove(3)
print(my_list)
```

आउटपुट(Output): [1, 2, 4]

10. रिव्हर्स reverse(): या मेथडचा उपयोग करून तुम्ही लिस्टमधील घटकाची उलट क्रमवारी लावू शकता.

```
my_list = [1, 2, 3, 4]
my_list.reverse()
print(my_list)
```

आउटपुट(Output): [4, 3, 2, 1]

11. क्रमाने लावणे sort(): या मेथडचा उपयोग करून तुम्ही लिस्टमधील घटकाची क्रमवारी लावू शकता.

```
my_list = [4, 2, 1, 3]
my_list.sort()
print(my_list)
```

आउटपुट(Output): [1, 2, 3, 4]

2.2 टपल्स (Tuples):

टपल्स (Tuples) पायथन मध्ये एक संगणक डेटा संरचना आहे ज्यात घटक अद्यातवत राखू शकतात आणि एकच क्षणी घटक न बदलता येतात. टपल्स अपरिवर्तनीय (immutable) आहे, असेही म्हणता की घटक त्याची मूल्ये बदलू शकतात, परंतु त्याच्या संख्येने या मूल्यांची कोणतीही बदल होत नाही . टपल्सचा वापर एकाच व्हेरिएबलमध्ये एकाधिक आयटम संचयित करण्यासाठी केला जातो.

टपल्सला परिभाषित करण्यासाठी (parentheses) आणि प्रत्येक एलेमेंट मध्ये कॉमा वापरून तयार केले जाते.

उदाहरणाथि, एक टपल्स तयार करूया:

```
my_tuple = (1, 2, 3, "Hello", 4.5)
print(my_tuple)
```

आउटपुट(Output): (1, 2, 3, 'Hello', 4.5)

टपल्समध्ये घटक एक वर्ण क्रमानुसार ठरतात, आणि तुम्ही टपल्स मध्ये इंडेक्स वापरून कोणत्याही घटकाला प्राप्त करू शकता.

```
print(my_tuple[0])
```

आउटपुट(Output): 1

```
print(my_tuple[3])
```

आउटपुट(Output): 'Hello'

टपल्स सुरक्षित आहे, असे म्हणता की टपल्स ची मूल्ये बदलली जाऊ शकत नाहीत. परंतु तुम्ही एक नवीन टपल्स तयार करू शकता

```
new_tuple = my_tuple + (6, 7)
```

```
print(new_tuple)
```

आउटपुट(Output): (1, 2, 3, 'Hello', 4.5, 6, 7)

टपल्स मध्ये विविध प्रकारचा डेटा टाइप्स समर्थन करता. उपयोगकर्त्यांना टपल्स चेअर्धकांश उपयोगन खासगी बाजूला सर्व काही घटक संगणकीय रूपात ठरविता येतात.

2.2.1 टपल्समधील व्हॅल्यू ऍक्सेस करणे (Accessing values in Tuples)

टपल्समध्ये घटक इंडेक्सिंग माध्यमात प्राप्त केली जातात. पहिल्या घटकासाठी इंडेक्स 0 पासून सुरु होतो.

```
my_tuple = (1, 2, 3, 'Hello', 4.5)
```

मूल्य प्राप्त करणे

```
print(my_tuple[0])
```

आउटपुट(Output): 1

```
print(my_tuple[3])
```

आउटपुट(Output): 'Hello'

2.2.2 टपल्स मधील व्हॅल्यू डिलीट करणे (deleting values in Tuples):

आधी तयार केलेले टपल्स बदलता येत नाही, तुम्ही प्रत्यक्षपणे टपल्स मध्ये घटक हटवू शकत नाही. परंतु, तुम्ही आपल्या आवडीच्या बदलांसह नवीन टपल्स तयार करू शकता.

```
my_tuple = (1, 2, 3, 'Hello', 4.5)
```

मूल्य हटविणे (नवीन टपल्स तयार करणे 'Hello' बाहेर काढून)

```
new_tuple = my_tuple[:3] + my_tuple[4:]
```

```
print(new_tuple)
```

आउटपुट(Output): (1, 2, 3, 4.5)

2.2.3 टपल्समधील मूल्य अपडेट करणे (updating Tuples)

टपल्स अपरिवर्तनीय असताना, तुम्ही प्रत्यक्षपणे किंवा स्थानिकपणे टपल्स ची व्हॅल्यू अपडेट करू शकत नाही. परंतु, तुम्ही आपल्या आवडीच्या बदलांसह नवीन टपल्स तयार करू शकता.

```
my_tuple = (1, 2, 3, 'Hello', 4.5)
```

मूल्य अद्यतन करणे(नवीन टपल्स तयार करणेसंपादनांसह)

```
updated_tuple = my_tuple[:3] + ('World') + my_tuple[4:]
```

```
print(updated_tuple)
```

आउटपुट(Output): (1, 2, 3, ' World', 4.5)

लक्षात घ्या संपादन केलेल्या टपल्स मध्ये बदल होत आहे किवा नाही हे दर्शवणार, परंतु टपल्स आहे असे ठरताना, तुम्ही संपादन केलेल्या टपल्स ची मूल्ये बदलू शकत नाहीत. स्थानिकपणे , टपल्स मध्ये नवीन घटक जोडता तुम्ही टपल्स ची मूल्ये बदलू शकता.

2.2.4 बेसिक टपल्स फंक्शन (Basic Tuple function):

- **काँकॅटिनेशन (Concatenation):** दोन ट्युपल्स एकत्र जोडण्यासाठी + ऑपरेटर वापरला जातो.
- **रिपीटेशन (Repetition):** ट्युपलच्या घटकांना * ऑपरेटर वापरून अनेक वेळा पुन्हा वापरले जाऊ शकते.
- **स्लायसिंग (Slicing):** ट्युपलच्या विशिष्ट भागाचा स्लाइस घेण्यासाठी : ऑपरेटर वापरला जातो.
- **मेम्बरशिप (Membership):** in कीवर्ड वापरून एखादा घटक ट्युपलमध्ये आहे की नाही हे तपासले जाते.
- **ईटिरेशन (Iteration):** for लूप वापरून ट्युपलमधील प्रत्येक घटकावर प्रक्रिया करता येते.
- **कंप्यारिजण (Comparison):** दोन ट्युपल्स एकसारखे आहेत की नाही हे तपासण्यासाठी == ऑपरेटर वापरला जातो.

Creating tuples

tuple1 = (1, 2, 3)

tuple2 = (4, 5, 6)

1. काँकॅटिनेशन (Concatenation)

concatenated = tuple1 + tuple2

print('Concatenated tuple: {concatenated}')

आउटपुट(Output): Concatenated tuple: (1, 2, 3, 4, 5, 6)

2. रिपीटेशन (Repetition)

repeated = tuple1 * 2

print('Repeated tuple: {repeated}')

आउटपुट(Output): Repeated tuple: (1, 2, 3, 1, 2, 3)

3. स्लायसिंग (Slicing)

sliced = concatenated[1:4]

print('Sliced tuple: {sliced}')

आउटपुट(Output): Sliced tuple: (2, 3, 4)

4. मेम्बरशिप (Membership)

is_member = 3 in tuple1

print('Is 3 in tuple1? {is_member}')

आउटपुट(Output): Is 3 in tuple1? True

5. ईटिरेशन (Iteration)

print('Elements in tuple1:')

for element in tuple1:

```
print(element)
```

आउटपुट(Output):

Elements in tuple1:

```
1
2
3
```

6. कंप्यारिजण (Comparison)

```
tuple3 = (1, 2, 3)
```

```
are_equal = tuple1 == tuple3
```

```
print ('Are tuple1 and tuple3 equal? {are_equal}')
```

आउटपुट(Output):

Are tuple1 and tuple3 equal?

```
True
```

2.3 सेट(Set):

पायथन मध्ये सेट्स म्हणजे एक अशा प्रकारचा डेटा स्ट्रक्चर आहे ज्यात पुन्हा न आलेले (unique) आणि बदल न करू शकणारे (immutable) घटक असतात. सेट्स असंघटित (unordered) असतात म्हणजे त्यामध्ये घटकांचे कोणतेही ठराविक क्रम (order) नसते.

2.3.1 सेट तयार करणे (Set Creation)

सेट तयार करण्यासाठी कर्ली ब्रेसिस { } किंवा set() फंक्शन वापरले जाते.

```
# Creating a set using curly braces
```

```
my_set = {1, 2, 3, 4, 5}
```

```
# Creating a set using the set() function
```

```
another_set = set([6, 7, 8, 9, 10])
```

```
# Printing the sets
```

```
print('My set: {my_set}') # Output: My set: {1, 2, 3, 4, 5}
```

```
print('Another set: {another_set}')
```

आउटपुट(Output): Another set: {6, 7, 8, 9, 10}

1. सेटमध्ये घटक प्राप्त करणे (Reading value from the set)

सेट अव्यवस्थित असल्यामुळे, त्यातील एका किंवा एकाआधिक घटक निर्धारित करता येत नाही. आपण घटक किंवा त्यांच्या स्थानानुसार वाचू शकता. परंतु, in कीवर्ड वापरून सध्या तो घटक त्या सेट मध्ये आहे कि नाही याची तपासणी करू शकता.

```
.my_set = {1, 2, 3, 4, 5}
```

3 सेटमध्ये आहे की नाही हे तपासा

```
is_present = 3 in my_set
```

2. सेटमधून घटक काढणे (Deleting Values from set)

सेटमधून घटक काढण्यासाठी खालील पद्धती वापरल्या जाऊ शकतात:

- `remove(element)`: दिलेला घटक काढून टाकतो. घटक सेटमध्ये नसल्यास की एरर (KeyError) दाखवतो.
- `discard(element)`: दिलेला घटक काढून टाकतो. घटक सेटमध्ये नसल्यास की एरर दाखवत नाही.
- `pop()`: सेटमधून एक रँडम घटक काढून टाकतो आणि त्याला परत करतो.
- `clear()`: सेट रिकामा करते.

```
my_set = {1, 2, 3, 4, 5}
```

```
# remove(element) - Remove an element (raises an error if element not found)
```

```
my_set.remove(3)
```

```
print (Set after removing 3: {my_set})
```

आउटपुट(Output):

```
Set after removing 3: {1, 2, 4, 5}
```

3. सेट अपडेट करणे (Updating Values in set)

सेट्समध्ये आधी असलेल्या घटकांमध्ये बदल करता येत नाही कारण सेट्समधील घटक अपरिवर्तनीय (immutable) असतात. मात्र, आपण सेटमध्ये नवीन घटक जोडू शकतो आणि आधीचे घटक काढू शकतो.

`add(element)`: सेटमध्ये नवीन घटक जोडतो.

`update(elements)`: सेटमध्ये एकापेक्षा अधिक घटक जोडता येतात.

```
# Creating a set
```

```
my_set = {1, 2, 3}
```

```
# add(element) - Add a single element to the set
```

```
my_set.add(4)
```

```
print('Set after adding 4: {my_set}')
```

आउटपुट(Output): Set after adding 4: {1, 2, 3, 4}

```
# update(elements) – Add multiple elements to the set
```

```
my_set.update([5, 6, 7])
```

```
print(Set after adding multiple elements: {my_set})
```

आउटपुट(Output):

```
Set after adding multiple elements:
```

```
{1, 2, 3, 4, 5, 6, 7}
```

2.3.2 बेसिक ऑपरेशन ऑन सेट (Basic Operations on Sets)

- **Union (संघ)**- दोन किंवा अधिक सेट्सचा संघ म्हणजे त्या सर्व सेट्समधील सर्व घटकांचा संघ.

```
Creating sets
```

```
set_a = {1, 2, 3}
```

```
set_b = {3, 4, 5}
```

```
# Union
```

```
union_set = set_a.union(set_b)
print(Union: {union_set})
```

आउटपुट(Output): Union: {1, 2, 3, 4, 5}

- **Intersection (छेद)**- दोन किंवा अधिक सेट्सचा छेद म्हणजे त्या सर्व सेट्समध्ये समान किंवा सारख्या असलेले घटकांचा संच

```
# Creating sets
set_a = {1, 2, 3}
set_b = {3, 4, 5}
# Intersection
intersection_set = set_a.intersection(set_b)
print(Intersection: {intersection_set})
```

आउटपुट(Output): Intersection: {3}

- **Difference (फरक)**- एका सेटमधून दुसऱ्या सेटमधील घटक वगळून उरलेला संच म्हणजे फरक.

```
# Creating sets
set_a = {1, 2, 3}
set_b = {3, 4, 5}
# Difference
difference_set = set_a.difference(set_b)
print('Difference: {difference_set}')
```

आउटपुट(Output): Difference: {1, 2}

2.3.3 बिल्ट-इन सेट फन्क्शन (Built-in Set functions):

1. len() - लांबी: हे एक सेटमध्ये असलेल्या मूल्यांची संख्या मिळवण्यासाठी वापरले जाते.

```
length = len(my_set)
```

2. add() – मूल्य जोडणे: हे सेटमध्ये एक मूल्य जोडते.

```
my_set.add(7)
```

3. remove() – मूल्य हटविणे: हे सेटमध्ये निर्देशित मूल्य हटवते. जर मूल्य अस्तित्वात नसतो तर हे एक KeyError उत्पन्न करते.

```
my_set.remove(4)
```

4. pop() - एक मूल्यहटवणे: हे सेटमध्ये कोणतेही एक मूल्य हटवते. जर सेट रर आहेतर हे एक KeyError उत्पन्न करते.

```
popped_value = my_set.pop()
```

5. clear() - सेट साफ करणे: हे सेटमध्ये मूल्य हटवते, त्यामुळे सेट रिक्त होते.

```
my_set.clear()
```

2.4 डिक्शनरी (Dictionaries):

पायथन मध्ये डिक्शनरी म्हणजे की-वॅल्यू जोड्यांचा (key-value pairs) संग्रह आहे. प्रत्येक की-वॅल्यू जोडीमध्ये की अद्वितीय असते आणि त्या कीसाठी संबंधित वॅल्यू असते. डिक्शनरी असंगठित (unordered) असते म्हणजे घटकांचे कोणतेही ठराविक क्रम (order) नसते. शब्दकोश हा एक संग्रह आहे जो ऑर्डर केलेला, बदलण्यायोग्य आहे आणि डुप्लिकेटला परवानगी देत नाही.

2.4.1 डिक्शनरी तयार करणे (Dictionary Creation)

डिक्शनरी तयार करण्यासाठी कर्ली ब्रेसेस { } किंवा dict() फंक्शन वापरले जाते.

Creating a dictionary using curly braces

```
my_dict = {
    'name': 'Alice',
    'age': 25,
    'city': 'Pune'
}
print(my_dict)
```

आउटपुट(Output):

```
name : Alice
age : 25
city: Pune
```

2.4.2 डिक्शनरी मधील घटक प्राप्त करणे (Accessing Values in Dictionary)

डिक्शनरीमधील व्हॅल्यू त्यांच्या की वापरून ऍक्सेस केल्या जातात.

Creating a dictionary using curly braces

```
my_dict = {
    'name': 'Alice',
    'age': 25,
    'city': 'Pune'
}

// Accessing values
name = my_dict['name']
age = my_dict.get('age')
city = my_dict.get('city', 'Unknown') # Default value if key is not found
print(Name: {name}) # Output: Name: Alice
print(Age: {age}) # Output: Age: 25
print(City: {city}) # Output: City: Pune
```

2.4.3 डिक्शनरीतील व्हॅल्यू डिलीट करणे (Deleting values in Dictionary)

डिक्शनरीमधून की-वॅल्यू जोड्या काढण्यासाठी काही पद्धती आहेत. खालील पद्धती वापरून आपण डिक्शनरीमधून घटक काढू शकतो:

- del कीवर्ड: विशेष की वापरून की-वॅल्यू जोड्या काढतो.
- pop() मेथड: विशेष की वापरून की-वॅल्यू जोड्या काढतो आणि वॅल्यू परत करतो.
- popitem() मेथड: शेवटची की-वॅल्यू जोडी काढतो आणि ती परत करतो.
- clear() मेथड: सर्व की-वॅल्यू जोड्या काढून डिक्शनरी रिकामी करते.

```
# Creating a dictionary
student_info = {
    'name': 'Rohit',
```

```
'age': 21,
'course': 'Computer Science'
}
```

Example 1: Using del() Method

```
del student_info['age']
print(After deleting age: {student_info})
```

आउटपुट(Output): After deleting age: {'name': 'Rohit', 'course': 'Computer Science'}

Example 2: Using pop() Method

```
# Creating a dictionary
employee_info = {
    'name': 'Meera',
    'position': 'Software Developer',
    'salary': 75000
}
# Deleting a key-value pair using pop()
salary = employee_info.pop('salary')
print('After popping salary: {employee_info}')
print('Popped salary: {salary}')
```

आउटपुट(Output):

After popping salary: {'name': 'Meera', 'position': 'Software Developer'}
Popped salary: 75000

2.4.4 डिक्शनरीतील व्हॅल्यू अपडेट करणे (Updating a Dictionary in Python)

पायथन मध्ये डिक्शनरीमधील व्हॅल्यूज अद्ययावत करण्यासाठी विविध पद्धती वापरता येतात. खालील पद्धती वापरून आपण डिक्शनरीमधील व्हॅल्यूज अद्ययावत करू शकतो:

1. Direct Assignment (थेट नियुक्ती): विशिष्ट कीसाठी व्हॅल्यू थेट बदलता येते.
2. update() Method (अपडेट मेथड): एकाच वेळी एकापेक्षा जास्त की-व्हॅल्यू जोड्या अद्ययावत करण्यासाठी वापरता येते.

Example 1: Using Direct Assignment

```
# Creating a dictionary
student_info = {
    'name': 'Rohit',
    'age': 21,
    'course': 'Computer Science'
}
# Updating a value using direct assignment
student_info['age'] = 22
student_info['course'] = 'Data Science'
print(Updated dictionary: {student_info})
```

आउटपुट(Output):

Updated dictionary: {'name': 'Rohit', 'age': 22, 'course': 'Data Science'}

2.4.5 बेसिक डिक्शनरी ऑपरेशन (Basic Dictionary Operations)

- डिक्शनरी तयार करणे: { } किंवा dict() वापरून.
- वॅल्यू ऍक्सेस करणे: [] किंवा get() वापरून.
- वॅल्यू अद्ययावत करणे: [] वापरून.
- नवीन की-वॅल्यू जोड्या जोडणे: [] वापरून.
- की-वॅल्यू जोड्या काढणे: del किंवा pop() वापरून.
- सर्व कीज मिळवणे: keys() मेथड वापरून.
- सर्व वॅल्यूज मिळवणे: values() मेथड वापरून.
- सर्व की-वॅल्यू जोड्या मिळवणे: items() मेथड वापरून.

2.4.6 बिल्ट-इन डिक्शनरी फंक्शन्स (Built-in Function in Dictionary)

पायथन मध्ये अंतर्भूत (बिल्ट-इन) डिक्शनरी फंक्शन्स वापरून, डिक्शनरीसह विविध ऑपरेशन्स कसे करायचे हे समजून घेऊया. Python मध्ये dict हा प्रकार (type) डिक्शनरीसाठी वापरला जातो. डिक्शनरी म्हणजे की (key) आणि वॅल्यू (value) यांच्या जोड्या असतात.

आता आपण काही महत्वाच्या बिल्ट-इन फंक्शन्स पाहूया:

- dict() - नवीन डिक्शनरी तयार करण्यासाठी.
- len() - डिक्शनरीतील की - वॅल्यू जोड्यांची संख्या शोधण्यासाठी.
- keys() - सर्व की मिळवण्यासाठी.
- values() - सर्व मूल्ये मिळवण्यासाठी.
- items() - सर्व की - वॅल्यू जोड्या मिळवण्यासाठी.
- get(key) - दिलेल्या की ची वॅल्यू मिळवण्यासाठी.
- update() - एक डिक्शनरी दुसऱ्या डिक्शनरीने अपडेट करण्यासाठी.
- pop(key) - दिलेली की व त्याचे वॅल्यू काढून टाकण्यासाठी.
- clear() - सर्व की - वॅल्यू जोड्या काढून टाकण्यासाठी.
- copy() - डिक्शनरीची प्रत तयार करण्यासाठी.

उदाहरणांसह समजून घेऊया:

```
# Create a new dictionary
example_dict = dict()
# Add some key-value pairs to the dictionary
example_dict['one'] = 1
example_dict['two'] = 2
example_dict['three'] = 3
print("Dictionary:", example_dict)
# Get the number of key-value pairs in the dictionary
```

```

print("Length:", len(example_dict))
# Get all the keys
print("All keys:", example_dict.keys())
# Get all the values
print("All values:", example_dict.values())
# Get all key-value pairs
print("All items:", example_dict.items())
# Get the value for a specific key
print("Value for key 'two':", example_dict.get('two'))
# Update the dictionary with another dictionary
example_dict.update({'four': 4, 'five': 5})
print("Updated dictionary:", example_dict)
# Remove a key-value pair
example_dict.pop('three')
print("Dictionary after removing 'three':", example_dict)
# Remove all key-value pairs
example_dict.clear()
print("Empty dictionary:", example_dict)
# Create a new dictionary and make a copy of it
example_dict = {'one': 1, 'two': 2, 'three': 3}
example_dict_copy = example_dict.copy()
print("Copied dictionary:", example_dict_copy)

```

2.5 युज ऑफ बिल्ट-इन पायथन फंक्शन्स (Use of Python Built-In Functions)

पायथन मध्ये अनेक इन-बिल्ट (मूलभूत) फंक्शन्स आहेत जी कोडिंग करताना खूप उपयुक्त ठरतात. खाली काही महत्वाच्या इन-बिल्ट फंक्शन्सची यादी आणि त्यांचे स्पष्टीकरण दिले आहे:

- len(): लांबी मिळवणे.
- type(): प्रकार मिळवणे.
- max() आणि min(): सर्वाधिक आणि सर्वात कमी मूल्य मिळवणे.
- sum(): जोड.
- sorted(): सॉर्ट करणे.
- abs(): मान मिळवणे.
- round(): गोल करणे.
- range(): श्रेणी तयार करणे.
- enumerate(): अनुक्रमांक आणि घटक जोडणे.
- zip(): जोड तयार करणे.
- map(): फंक्शन लागू करणे.
- filter(): घटक फिल्टर करणे.

पायथन मध्ये डेटा कन्व्हर्जन आणि गणितीय फंक्शन्ससाठी विविध इन-बिल्ट फंक्शन्स उपलब्ध आहेत. खालील उदाहरणे या फंक्शन्सची स्पष्टीकरण देतात.

2.5.1 डेटा कन्व्हर्जन फंक्शन्स (Data Conversion Functions)

1. **int()** : दिलेली संख्या किंवा स्ट्रिंगला पूर्णांकात (integer) रूपांतरित करण्यासाठी वापरले जाते.

```
# Converting string to integer
number_str = "123"
number_int = int(number_str)
print('Integer: {number_int}, Type: {type(number_int)}')
```

Output: Integer: 123, Type: <class 'int'>

2. **float()**: दिलेला डेटा फ्लोटिंग पॉइंट (decimal) मध्ये रूपांतरित करण्यासाठी वापरले जाते.

```
# Converting string to float
number_str = "123.45"
number_float = float(number_str)
print('Float: {number_float}, Type: {type(number_float)}')
```

Output: Float: 123.45, Type: <class 'float'>

3. **list()** - दिलेला डेटा लिस्टमध्ये (list) मध्ये रूपांतरित करण्यासाठी वापरले जाते.

```
# Converting string to list
text = "hello"
text_list = list(text)
print('List: {text_list}, Type: {type(text_list)}')
```

Output: List: ['h', 'e', 'l', 'l', 'o'], Type: <class 'list'>

2.5.2 मॅथ फंक्शन्स (Math Functions)

1. **abs()** -संख्येचे मान (absolute value) मिळवण्यासाठी वापरले जाते.

```
# Getting the absolute value
negative_number = -10
absolute_value = abs(negative_number)
print('Absolute Value: {absolute_value}')
```

Output: Absolute Value: 10

2. **round()** - अपूर्णांक संखेला राऊंड ऑफ करण्यासाठी या मेथोड चा वापर केला जातो.

```
# Rounding off the number
number = 4.678
rounded_number = round(number, 2)
print('Rounded Number: {rounded_number}')
```

Output: Rounded Number: 4.68

3. **max()** आणि **min()** -मूल्यांचे कमाल (max()) आणि किमान (min()) मूल्य शोधण्यासाठी या मेथोडचा वापर केला जातो.

```
# Creating a list of numbers
numbers = [3, 5, 7, 2, 8]
```

```
# Using max() to find the maximum value
maximum = max(numbers)
print('Maximum: {maximum}')
```

Output: Maximum: 8

```
# Using min() to find the minimum value
minimum = min(numbers)
print('Minimum: {minimum}')
```

Output: Minimum: 2

4. **sum()** - संख्येची बेरीज करण्यासाठी वापरले जाते.

```
# Creating a list of numbers
numbers = [3, 5, 7, 2, 8]
# Using sum() to get the sum of numbers
total = sum(numbers)
print('Total: {total}')
```

Output: Total: 25

5. **pow()**- संख्येचे घातांक (power) शोधण्यासाठी वापरले जाते.

```
# Using pow( ) to calculate power
base = 2
exponent = 3
power = pow(base, exponent)
print('Power: {power}')
```

Output: Power: 8

2.6 युझर डिफाईन फंक्शन (User Defined Function)

पायथनमध्ये, यूजर-डिफाईन्ड फंक्शन म्हणजे एक असे प्रोग्रामिंग ब्लॉक असते ज्याला यूजरने डिफाइन केलेले असते. यामुळे कोड पुन्हा वापरणे आणि प्रोग्रामिंग अधिक कार्यक्षम बनवणे शक्य होते. पायथनमध्ये फंक्शन डिफाइन करण्यासाठी def कीवर्ड वापरला जातो. खालीलप्रमाणे एक फंक्शन डिफाइन केले जाते:

2.6.1 फंक्शनचे महत्व (Importance of function):

- **कोड पुनर्वापर:** एकदा फंक्शन लिहून ते अनेक वेळा वापरता येते.
- **विभाजन आणि संगणन:** मोठे प्रोग्राम्स लहान, व्यवस्थापित करण्यायोग्य फंक्शन्समध्ये विभागता येतात.
- **सपष्टता आणि वाचनयोग्यता:** फंक्शन्समुळे कोड अधिक सपष्ट आणि वाचायला सोपा होतो.
- **डीबगिंग आणि चाचणी:** फंक्शन्स स्वतंत्रपणे डीबग आणि चाचणी करता येतात.

2.6.2 युझर डिफाईन फंक्शन चे महत्वाचे घटक(Important element of User define function):

1. फंक्शनची व्याख्या (Function Definition) - def कीवर्ड वापरून.
2. फंक्शन कॉलिंग(Function Calling) - फंक्शनचे नाव आणि पॅरामीटर्स वापरून.
3. पॅरामीटर्स आणि आर्ग्युमेंट्स (Parameters and Arguments) - फंक्शनला पॅरामीटर्स पास करता येतात आणि कॉल करताना आर्ग्युमेंट्स देता येतात.

4. रिटर्न स्टेटमेंट (Return Statement) - फंक्शनला मूल्य परत करण्यासाठी वापरले.

- **फंक्शनची मूलभूत रचना (Syntax of function):**

```
def function_name(parameters):
```

```
    """
```

```
    Docstring: फंक्शन काय करते याचे वर्णन
```

```
    """
```

```
    # फंक्शन बॉडी
```

```
    # कोड इथे येतो
```

```
    return result # परिणामी मूल्य (optional)
```

उदाहरणे:

1. साधे फंक्शन (Simple Function): हे फंक्शन दोन संख्यांचा बेरिज करते.

```
def add(a, b):
```

```
    """
```

```
    दोन संख्यांचा बेरिज करणारे फंक्शन
```

```
    """
```

```
    return a + b
```

```
# फंक्शन कॉल
```

```
result = add(3, 5)
```

```
print(result)
```

```
# Output: 8
```

Another Example-

```
# Defining a function
```

```
def greet():
```

```
    print("Hello, welcome to Python!")
```

```
# Function calling
```

```
greet()
```

```
# Output: Hello, welcome to Python!
```

2. डिफॉल्ट पॅरामीटर्ससह फंक्शन (Function with Default parameter): हे फंक्शन दोन संख्यांचा बेरीज करते आणि दुसऱ्या संख्येची डिफॉल्ट व्हॅल्यू 0 ठेवते.

```
def add(a, b=0):
```

```
    """
```

```
    दोन संख्यांचा बेरीज करणारे फंक्शन, जिथे दुसऱ्या संख्येची डिफॉल्ट व्हॅल्यू 0 आहे.
```

```
    """
```

```
    return a + b
```

```
# फंक्शन कॉल्स
```

```
result1 = add(3)
```

```
result2 = add(3, 5)
```

```
print(result1) # Output: 3
```

```
print(result2) # Output: 8
```

3. कीवर्ड आर्गुमेंट्ससह फंक्शन(Function with arguments): हे फंक्शन कीवर्ड आर्गुमेंट्स वापरून कॉल केले जाते.

```
def add(a, b):
    """
    दोन संख्यांचा बेरीज करणारे फंक्शन
    """
    return a + b
```

कीवर्ड आर्गुमेंट्ससह फंक्शन कॉल

```
result = add(a=3, b=5)
print(result) # Output: 8
```

4. अनेक पॅरामीटर्ससह फंक्शन (*args, **kwargs): *args म्हणजे अनिश्चित संख्येचे आर्गुमेंट्स आणि **kwargs म्हणजे कीवर्ड आर्गुमेंट्स.

```
def add(*args):
    """
    अनेक संख्यांचा बेरीज करणारे फंक्शन
    """
    return sum(args)
```

फंक्शन कॉल्स

```
result1 = add(1, 2, 3)
result2 = add(4, 5, 6, 7)
print(result1) # Output: 6
print(result2) # Output: 22
```

5. रिटर्न स्टेटमेंट (Return Statement) - रिटर्न स्टेटमेंट वापरून फंक्शन कसे मूल्य परत करते.

Defining a function with return statement

```
def multiply_numbers(a, b):
    return a * b
```

Function calling and storing the result

```
product = multiply_numbers(4, 3)
print("Product: {product}")
```

Output: Product: 12

2.6.3 स्कोप ऑफ व्हेरिबल इन पायथन (Scope of Variables in Python)

पायथन मध्ये, व्हेरिबल्सची स्कोप म्हणजे त्यांची वैधता आणि प्रवेशयोग्यता याचे क्षेत्र. याचे मुख्य दोन प्रकार आहेत:

- Local Scope (लोकल स्कोप)
- Global Scope (ग्लोबल स्कोप)

1. Local Scope (लोकल स्कोप)

लोकल स्कोप म्हणजे फंक्शनच्या आत तयार केलेले व्हेरिबल्स फक्त त्या फंक्शनच्या आतूनच प्रवेशयोग्य असतात. हे व्हेरिबल्स फंक्शन संपल्यावर नष्ट होतात.

```
def local_scope_example():
```

```
    local_var = "I am local"
```

```
    print(local_var)
```

```
local_scope_example ()
```

```
# Output: I am local
```

```
# Trying to access local_var outside the function will cause an error
```

```
# print(local_var) # This will cause a NameError
```

2. Global Scope (ग्लोबल स्कोप) -ग्लोबल स्कोप म्हणजे फंक्शनच्या बाहेर तयार केलेले व्हेरिएबल्स जे संपूर्ण प्रोग्राममध्ये प्रवेशयोग्य असतात. हे व्हेरिएबल्स स्क्रिप्ट संपल्यापर्यंत अस्तित्वात राहतात.

```
global_var = "I am global"
```

```
def global_scope_example( ):
```

```
    print(global_var)
```

```
global_scope_example()
```

```
# Output: I am global
```

```
# Accessing global_var outside the function is allowed
```

```
print(global_var)
```

```
# Output: I am global
```

- **मॉडीफायिंग ग्लोबल व्हेरिएबल इन्साईड फंक्शन (Modifying Global Variable inside Function)**

फंक्शनच्या आत ग्लोबल व्हेरिएबल बदलण्यासाठी global कीवर्ड वापरला जातो.

```
counter = 0
```

```
def increase_counter():
```

```
    global counter
```

```
    counter += 1
```

```
increase_counter ()
```

```
print(counter)
```

```
# Output: 1
```

- **एन्क्लोजिंग स्कोप (Enclosing Scope)**

फंक्शनच्या आत तयार केलेले व्हेरिएबल्स इतर फंक्शनच्या आतपासून प्रवेशयोग्य असतात. याला नॉन-लोकल स्कोप म्हणतात.

```
def outer_function( ):
```

```
    outer_var = "I am outer"
```

```
    def inner_function():
```

```
        nonlocal outer_var
```

```
        outer_var = "I have been changed"
```

```
        print(outer_var)
```

```
    inner_function ( )
```

```
    print(outer_var)
```

```
outer_function()
```

Output:

I have been changed

I have been changed

संदर्भसूची :

1. Codecademy: <https://www.codecademy.com/learn/learn-python>
2. Coursera: <https://www.coursera.org/>
3. edX: <https://www.edx.org/>
4. Python.org: <https://www.python.org/about/gettingstarted/>
5. W3Schools: <https://www.w3schools.com/python>

युनिट- 3

पायथॉन मॉड्युल्स आणि पॅकेज (Python, Modules and Packages)

विषय निष्पत्ती (Course Outcome):

दिलेल्या समस्येचे निराकरण करण्यासाठी पायथॉनमध्ये मॉड्यूल पॅकेजेस लागू करा.

घटक निष्पत्ती (Theory Learning Outcome):

1. दिलेल्या समस्येसाठी पायथन मॉड्यूल तयार करा.
2. दिलेल्या समस्येसाठी पायथन पॅकेज विकसित करा.
3. अरेवर गणिती क्रिया करण्यासाठी Numpy वापरा.

3.1 पायथॉन मॉड्युल्स (Python Modules)

3.1.1 पायथॉन मॉड्युल्स तयार करणे (Writing Python Modules)

मॉड्यूल म्हणजे काय? कोड लायब्ररीसारखेच मॉड्यूल वापरले जाते.

मॉड्युल्स म्हणजे तुम्हाला तुमच्या ॲप्लिकेशन मध्ये समाविष्ट करण्याच्या फंक्शनचा संच असलेली फाइल. मॉड्यूल तयार करण्यासाठी फाईल एक्स्टेंशन .py सह फाईलमध्ये तुम्हाला हवा असलेला कोड जतन करा: खाली दाखवलेली पायथन स्क्रिप्ट आहे.

ज्यामध्ये SayHello() फंक्शनची व्याख्या आहे. हे hello.py म्हणून सेव्ह केले आहे.

(Example) hello.py

```
def SayHello(name):
    print("Hello {}! How are you?".format(name))
    return
```

3.1.2 मॉड्युल्स आयात करणे (importing modules)

मॉड्युल्स आयात करणेसाठी import कीवर्ड वापरा

```
>>> import hello
>>> hello.SayHello("purva")
```

Output : Hello purva! How are you?

3.1.3 मॉड्युल्समधून ऑब्जेक्ट इम्पोर्ट करणे (importing objects from modules)

मॉड्यूलमधून फक्त भाग इम्पोर्ट करण्यासाठी from कीवर्ड वापरा

Example The module named mymodule has one function and one dictionary:

```
def greeting(name):
    print("Hello, " + name)
person1 = { "name": "John", "age": 36, "country": "Norway" }
Import only the person1 dictionary from the module:
from mymodule import person1
print (person1["age"])
```

Output: 36

3.1.4 मॉड्यूलला पुन्हा नाव देणे(Re-naming a Module) :-

जेव्हा आपण मॉड्यूल ऑब्जेक्ट इम्पोर्ट करता तेव्हा कीवर्ड `as` वापरून आपण सबनेम तयार करू शकता: (You can create an alias when you import a module, by using the `as` keyword).

```
import mymodule as mx
a=mx.person1["age"] print(a)
```

Output: 36

3.1.5 पायथॉन मधील बिल्ट इन मॉड्युल्स, (न्यूमरिकल आणि मॅथेमॅटिकल मॉड्यूल, फंक्शनल, प्रोग्रामिंग मॉड्यूल (Python built in modules, Numeric and Mathematical module, Functional Programming Module): -

पायथॉन मधील अंगभूत मॉड्युल्स :- पायथनमध्ये अनेक बिल्ट इन मॉड्युल्स आहेत, जे तुम्ही तुम्हाला हवे तेव्हा `import` कीवर्ड वापरून आयात करू शकता. आणि त्या मॉड्युल्स मध्ये उपलब्ध असलेल्या मेथड्स डॉट ऑपरेटर च्या साहाय्याने खालील दिलेल्या उदाहरणामध्ये नमूद केल्याप्रमाणे वापरू शकता. न्यूमरिकल आणि मॅथेमॅटिकल (अंकीय आणि गणितीय) ऑपरेशन करण्यासाठी पायथन खालील बिल्ट इन मॉड्यूल प्रदान करतो:

1. डेसिमल मॉड्यूल (Decimal module) Python त्याच्या पररभाषेत " डेसिमल " मॉड्यूल वापरून जलद डेसिमल पॉइंट अंकगणित करण्यासाठी विशिष्ट पद्धती प्रदान करते. डेसिमल मॉड्यूल(Decimal module) मधील `ln()` आणि `log10()` method

`ln()` :- हे फंक्शन डेसिमल संख्येच्या नैसर्गिक(natural) लॉगरिथम ची गणना करण्यासाठी वापरले जाते.

`log10()` :- हे फंक्शन डेसिमल संख्येच्या लॉग(बेस 10) ची गणना करण्यासाठी वापरले जाते.

`ln` आणि `log10()` च्या कार्याची प्रदर्शन करण्यासाठी पायथन कोड उदाहरण

Example

```
# importing "decimal" module to use decimal functions
import decimal
# using ln() to compute the natural log of decimal number
a = decimal.Decimal(4.5).ln()
# using sqrt() to compute the log10 of decimal number
b = decimal.Decimal(4.5).log10()
# printing the natural logarithm
print ("The natural logarithm of decimal number is : ",end="")
print (a)
# printing the log10
print ("The log(base 10) of decimal number is : ",end="")
print (b)
```

Output :- The natural logarithm of decimal number is : 1.504077396776274073373258352

The log(base 10) of decimal number is : 0.6532125137753436793763169118

2. अपूर्णांक मॉड्यूल (Fractions Module) हे मॉड्यूल परिमेय संख्या अंक गणितासाठी समर्थन प्रदान करते. हे पूर्णांक, फ्लोट्स, संख्या, दशांश आणि स्क्रॅंगमधून अपूर्णांक उदाहरण तयार करण्यास अनुमती

देते. अपूर्णांक उदाहरणे : पूर्णांकांच्या जोडीतून, दुसऱ्या परिमेय संख्येवरून किंवा स्ट्रिंग वरून अपूर्णांकाची उदाहरणे तयार केली जाऊ शकतात.

उदाहरण (Example) :-

```
from fractions
import Fraction
print (Fraction(11, 35)) # returns Fraction(11, 35)
print (Fraction(10, 18)) # returns Fraction(5, 9)
print (Fraction()) # returns Fraction(0, 1)
```

Output :- 11/ 5/9 0

3. गणित मॉड्यूल (Python - Math Module) :- पायथनमध्ये एक बिल्ट इन मॉड्यूल आहे जे तुम्ही गणिताच्या कामांसाठी वापरू शकता. गणित मॉड्यूलमध्ये पद्धती आणि स्थिरांकाचा संच असतो.

उदाहरण (Examples)

```
>>> import math
>>> math.pi 3.141592653589793
>>> math.log(10) 2.302585092994046
>>> math.sin(0.5235987755982988) 0.49999999999999994
>>> math.cos(0.5235987755982988) 0.8660254037844387
>>> math.tan(0.5235987755982988) 0.5773502691896257
>>> math.radians(30) 0.5235987755982988
>>> math.degrees(math.pi/6) 29.999999999999996
```

उदाहरण 1(Examples)

```
#Import math library
import math
#Round a number upward to its nearest integer
print(math.ceil(1.4))
print(math.ceil(5.3))
print(math.ceil(-5.3))
print(math.ceil(22.1))
print(math.ceil(10.1))
```

Output: 2 6 -5 23 11

उदाहरण 2 (Examples)

```
# Import math Library
import math
# Return the tangent of different numbers
print (math.tan(90))
print (math.tan(-90))
print (math.tan(45))
print (math.tan(60))
```

Output: -1.995200412208242 1.995200412208242 1.6197751905438615 .320040389379563

4. इटरटूल्स मॉड्यूल (Python's Itertools module)

हे एक मॉड्यूल आहे जे जटिल पुनरावृत्ती तयार करण्यासाठी पुनरावृत्तांवर कार्य करणारे विविध कार्ये प्रदान करते. हे मॉड्यूल एक जलद, मेमरी- कार्यक्षम साधन म्हणून कार्य करते जे एकतर स्वतः द्वारे किंवा पुनरावृत्ती बीज गणित तयार करण्यासाठी एकत्रित पणे वापरले जाते.

• इटरटूल्स मॉड्यूल मेथड्स (Itertools module methods)

count(start, step): Method Count (प्रारंभ, चरण):

हा पुनरावृत्तीकर्ता "प्रारंभ" क्रमकापासून मुद्रण सुरू करतो आणि अमर्यादप्रमाणे प्रिंट करतो. जर steps उल्लेख केला असेल, तर संख्या वगळली जाईल अन्यथा step 1 बाय डीफॉल्ट आहे.

```
from itertools import count
for number in count(start=1, step=2):
    if number > 10:
        break
    print(number) # print statement
```

Output : 1 3 5 7 9

• पुनरावृत्ती करता येण्याजोगा (cycle method) :

हा पुनरावृत्तीकर्ता पास केलेल्या कंटेनरमधून क्रमाने सर्व मूल्य मुद्रित करतो. जेव्हा सर्व घटक चक्रीय पद्धतीने मुद्रित केले जातात तेव्हा ते सुरवातीपासून पुन्हा मुद्रण सुरू करते.

Python program to demonstrate # infinite iterators

```
import itertools
count = 0
# for in loop
for i in itertools.cycle('AB'):
    if count > 7:
        break
    else:
        print(i, end=" ")
    count += 1
```

Output : - A B A B A B A B

5 . ऑपरेटर मॉड्यूल(operator module)

मॉड्यूल ऑपरेटरमध्ये अंतर्गत अनेक गणितीय, रीलेशनल, बीटवाइज इत्यादी ऑपरेशन्ससाठी पायथॉनमध्ये पूर्वनिर्धारित कार्ये आहेत.

• ऑपरेटर मॉड्यूल मेथड्स (Methods of Operator module):-

add(a, b):- हे फंक्शन दिलेल्या अर्ग्युमेंट्स ची ऍडिशन करते.

ऑपरेशन - a+ b.

sub(a, b):- हे फंक्शन दिलेल्या अर्ग्युमेंट्स मधील डिफरन्स (difference) दाखवते.

ऑपरेशन - a – b.

`mul(a, b)`:- हे फंक्शन दिलेल्या अर्ग्युमेंट्स चे प्रॉडक्ट (product) करते.

ऑपरेशन – `a * b`.

`truediv(a,b)`:- हे फंक्शन दिलेल्या अर्ग्युमेंट्स चे विभाजन मिळवून देते.

ऑपरेशन - `a / b`.

`floordiv(a,b)` :- हे फंक्शन दिलेल्या अर्ग्युमेंट्स चे विभाजन देखील देते. पण मूल्य फ्लोट व्हॅल्यू आहे म्हणजेच सर्वात मोठी लहान पूर्णांक मिळवते.

ऑपरेशन - `a // b`.

`pow(a,b)`):- हे फंक्शन दिलेल्या अर्ग्युमेंट्सचे घातांक मिळवते.

ऑपरेशन – `a ** b`.

`mod(a,b)` हे फंक्शन दिलेल्या अर्ग्युमेंट्स चे मॉड्यूलस परत करते.

ऑपरेशन - `a % b`.

add, sub, mul फंक्शन दर्शविण्यासाठी पायथन प्रोग्राम:

```
import operator
# Initializing variables
a = 4
b = 3
# using add() to add two numbers
print ("The addition of numbers is :",end="");
print (operator.add(a, b))
# using sub() to subtract two numbers
print ("The difference of numbers is :",end="");
print (operator.sub(a, b))
# using mul() to multiply two numbers
print ("The product of numbers is :",end="");
print (operator.mul(a, b))
```

Output:- The addition of numbers is:7

The difference of numbers is :1

The product of numbers is:12

truediv, floordiv, pow, mod फंक्शन दर्शविण्यासाठी पायथन प्रोग्राम:

```
import operator
# Initializing variables
a = 5 b = 2
# using truediv() to divide two numbers
print ("The true division of numbers is : ",end="");
print (operator.truediv(a,b))
# using floordiv() to divide two numbers
print ("The floor division of numbers is : ",end="");
print (operator.floordiv(a,b))
```

```
# using pow() to exponentiate two numbers
print ("The exponentiation of numbers is : ",end="");
print (operator.pow(a,b))
# using mod() to take modulus of two numbers
print ("The modulus of numbers is : ",end="");
print (operator.mod(a,b))
```

Output :- The true division of numbers is: 2.

The floor division of numbers is: 2

The exponentiation of numbers is: 25

The modulus of numbers is: 1

6. रँडम मॉड्यूल (Random Module) :- Python Random मॉड्यूल हे Python मध्ये रँडम पूर्णांक तयार करण्यासाठी अंगभूत मॉड्यूल आहे. या संख्या रँडम येतात आणि कोणत्याही नियम किंवा सूचनाचे पालन करत नाहीत. म्हणून आम्ही या मॉड्यूलचा वापर रँडम संख्या तयार करण्यासाठी, लिस्ट किंवा स्ट्रींगसाठी रँडम आयटम प्रदर्शित करण्यासाठी करू शकतो.

• **रँडम मॉड्यूल मेथड्स (Random Module methods) :-**

a. random() फंक्शन : random.random() फंक्शन 0.0 ते 1.0 पर्यंतचा फ्लोट क्रमांक देते. या कार्यासाठी कोणतेही पॅरामीटर आवश्यक नाहीत. ही पद्धत [०.० आणि १] मधील दुसरे रँडम फ्लोटिंग-पॉइंट मूल्य देते.

Python program for generating random float number

```
import random
```

```
num=random.random() print(num)
```

output : 0.3232640977876686

b. randint() फंक्शन : random.randint() फंक्शन पुरवलेल्या संख्यांच्या श्रेणीतून एक रँडम पूर्णांक तयार करते.

Python program for generating a random integer

```
import random
```

```
num = random.randint(1, 500)
```

```
print( num )
```

output: 215

c. randrange() फंक्शन : random.randrange()

फंक्शन स्टार्ट, स्टॉप आणि step पॅरामीटर द्वारे परिभाषित केलेल्या दिलेल्या रेंजमधून रँडम आयटम निवडते. डीफॉल्टनुसार, प्रारंभ 0 वर सेट केला जातो. त्याचप्रमाणे, step डीफॉल्टनुसार 1 वर सेट केली जाते.

To generate value between a specific range

```
import random
```

```
num = random.randrange(1, 10)
```

```
print( num )
```

```
num = random.randrange(1, 10, 2)
```

```
print( num )
```

output : 4 9

d. choice() फंक्शन :- random.choice() फंक्शन निर्दिष्ट क्रमातून randomly निवडलेला घटक परत करतो

```
# To select a random element
import random
random_s = random.choice('Random Module')
#a string
print( random_s )
random_l = random.choice([23, 54, 765, 23, 45, 45]) #a list
print( random_l )
random_s = random.choice((12, 64, 23, 54, 34)) #a set
print( random_s )
```

Output: M 765 54

• **फंक्शनल प्रोग्रामिंग मॉड्यूल (Functional programming modules) :-**

1. लॅम्बडा एक्सप्रेशन-लॅम्बडा फंक्शन (lambda function):- हे एक लहान निनावी फंक्शन आहे. लॅम्बडा फंक्शन कितीही अर्ग्युमेंट्स घेऊ शकतो, परंतु केवळ एक एक्सप्रेशन असू शकते. जेव्हा तुम्ही दुसऱ्या फंक्शनमध्ये निनावी फंक्शन म्हणून वापरता तेव्हा लॅम्बडाची शक्ती अधिक चांगली दर्शविली जाते.

Syntax : lambda arguments : expression

Example 1 :

```
x = lambda a : a + 10 print(x(5)) # Add 10 to argument a, and return the result:
```

Output : 15

Example 2 :

```
x = lambda a, b : a * b print(x(5, 6)) # Multiply argument a with argument b and return the result:
```

Output : 30

Example 3 :

```
a = lambda x, y, z : (x + y + z)
print(a(4, 5, 5))
```

Output : 14

2. map() फंक्शन :- हे फंक्शन लिस्ट मधील सर्व घटकांसाठी कार्यान्वित केले जाते. आणि प्रत्येक आयटमसाठी दिलेल्या फंक्शनद्वारे एक नवीन लिस्ट तयार केली जाते.

Example 1 :

```
def myfunc(n):
    return len(n)
x = map(myfunc, ('apple', 'banana', 'cherry'))
```

output : 5 5 6

Example 2 :

```
def addition(n):
```

```

return n + n # We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(addition, numbers)
print(list(result))

```

Output : [2, 4, 6, 8]

3. filter() फंक्शन : filter() फंक्शन एक पुनरावृत्ती करणारा परत करतो जेथे आयटम स्वीकारले आहे की नाही हे तपासण्यासाठी फंक्शनद्वारे आयटम फिल्टर केले जातात. filter() फंक्शनच्या मदतीने दिलेला क्रम फिल्टर करते जे अनुक्रमातील प्रत्येक घटक सत्य आहे की नाही याची टेस्ट करते.

Syntax filter(function, iterable)

Example 1 :

```

ages = [5, 12, 17, 18, 24, 32]
def myFunc(x):
    if x < 18:
        return False
    else:
        return True
adults = filter(myFunc, ages)
for x in adults: print(x)

```

Output : 18 24 36

Example 2 : # function that filters vowels :

```

def fun(variable):
    letters = ['a', 'e', 'i', 'o', 'u']
    if (variable in letters):
        return True
    else: return False
sequence = ['g', 'e', 'e', 'j', 'k', 's', 'p', 'r']
# using filter function
filtered = filter(fun, sequence)
print("The filtered letters are:")
for s in filtered: print(s)

```

output :- e e

4. reduce() function :- Python मध्ये, reduce() हे **बिल्ट इन** फंक्शन आहे जे पुनरावृत्ती करण्यायोग्य घटकांना दिलेला फंक्शन लागू करते, त्यांना एका मूल्यापर्यंत कमी करते.

The syntax for reduce() is as follows:

functools.reduce(function, iterable[, initializer])

Example :-

```

from functools import reduce
def add(a, b):
    return a + b
# Our Iterable

```

```
num_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

add function is passed as the first argument, and num_list is passed as the second argument

```
sum = reduce(add, num_list)
```

```
print("Sum of the integers of num_list : {sum}")
```

output : Sum of the integers of num_list : 55

3.2 पायथन पॅकेजेस (Python packages) :-

पायथनमध्ये फाइल्ससाठी डिरेक्टरी आणि मॉड्यूलसाठी पॅकेजेस आहेत. डिरेक्टरी मध्ये सब- डिरेक्टरीज आणि फाइल असू शकतात. पायथन पॅकेजमध्ये उप-पॅकेज आर्ग मॉड्यूल असू शकतात.

पायथनला पॅकेज म्हणून विचारात घेण्यासाठी निर्देशिकेत `__init__.py` नावाची फाइल असणे आवश्यक आहे. ही फाइल ब्लॉक असू शकते परंतु आम्ही सामान्यतः या फाइलमध्ये त्या पॅकेजसाठी इनिशलाइझेशन कोड ठेवतो.

3.2.1 पायथन पॅकेजेस लिहिणे (Writing Python packages)

पायथन पॅकेजेस लिहिण्यासाठी खालील स्टेप्स आहे :-

1. प्रथम, आम्ही एक डिरेक्टरी तयार करतो आणि त्यास शक्यतो त्याच्या ऑपरेशनशी संबंधित पॅकेजचे नाव देतो.
2. मग आम्ही त्यात क्लास आणि आवश्यक फंक्शन ठेवतो.
3. शेवटी डिरेक्टरी एक पॅकेज आहे हे पायथनला कळवण्यासाठी आम्ही डिरेक्टरी मध्ये `__init__.py` फाइल तयार करतो.

Example :

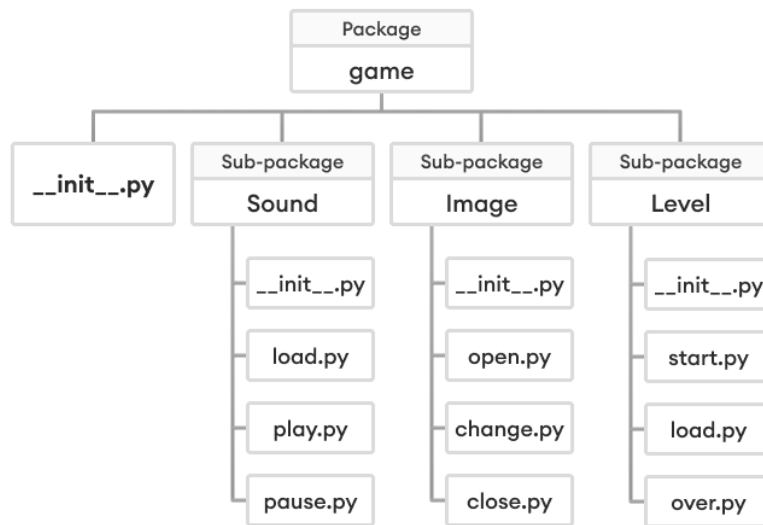


Figure. 3.2 Python Package Example

1. प्रथम फोल्डर गेम(game) तयार करा.
2. त्याच्या आत पुढा फोल्डर साउंड (sound) तयार करा.
3. साउंड फोल्डरच्या आत load.py फाइल तयार करा.

4. आत साउंड फोल्डर pause.py फाइल तयार करा.
5. आत साउंड फोल्डर play.py फाइल तयार करा.
6. पॅकेज गेम(game) आर्ण सबपॅकेज साउंड इम्पोर्ट करा (फाईल्स: लोड, पॉज, प्ले)

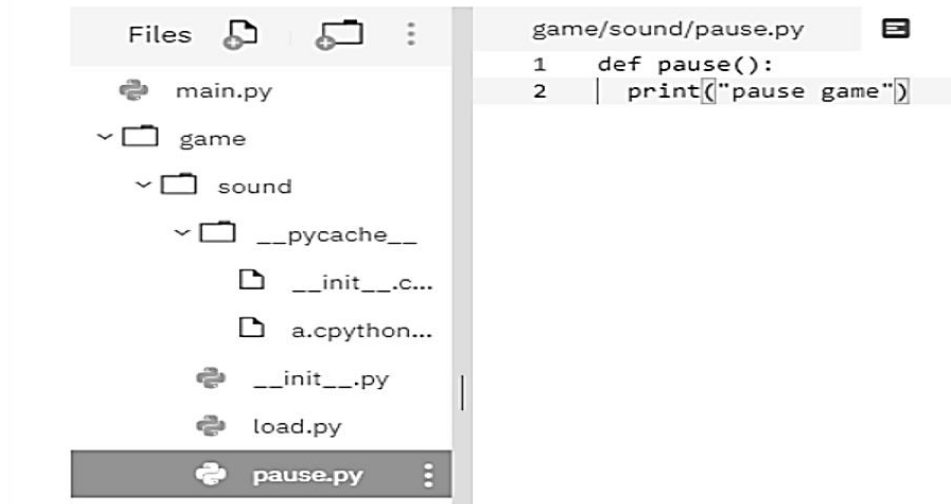


Figure 3.2.1: Step 1

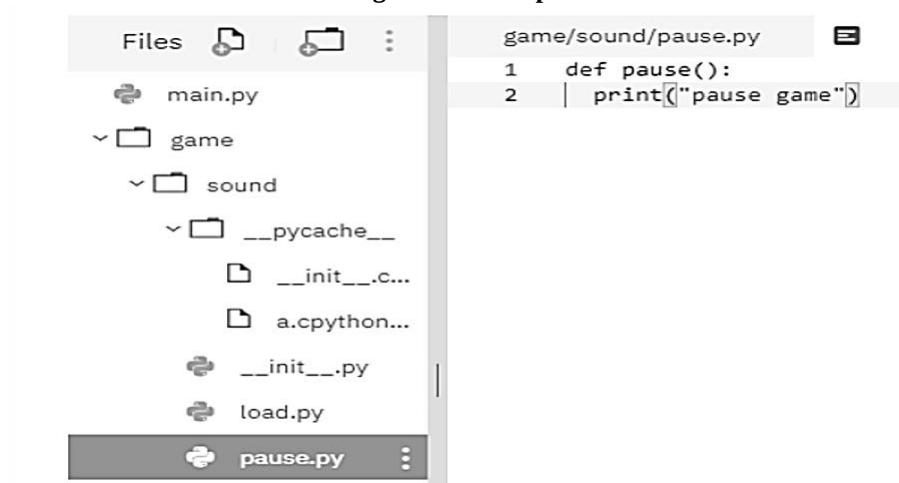


Figure 3.2.2: Step 2

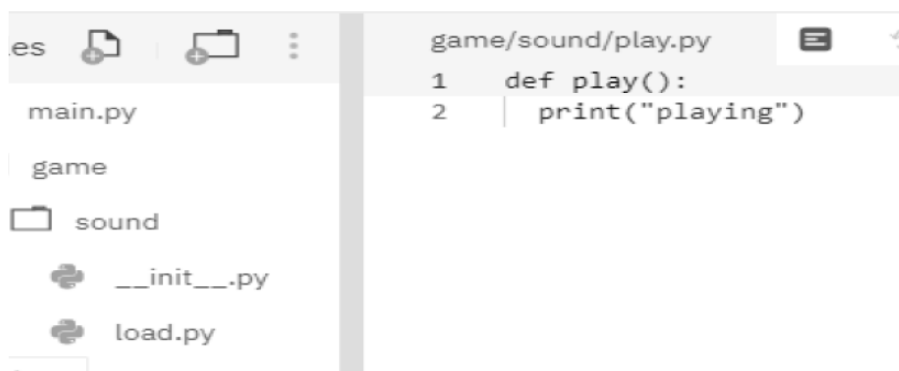


Figure 3.2.3: Step 3

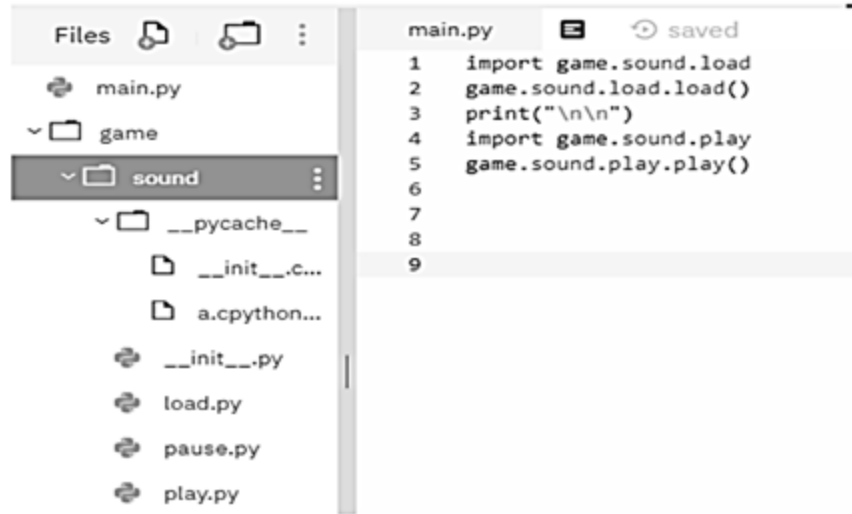


Figure 3.2.4: Step 4

Output :

loading data

playing

3.3 Numpy पॅकेजेस वापरणे Using Standard Numpy packages

NumPy ही एक पायथन लायब्ररी आहे जी **array** सोबत काम करण्यासाठी वापरली जाते.

NumPy म्हणजे **Numerical Python** संख्यात्मक पायथन.

NumPy का वापरावे?

1. Python मध्ये आमच्याकडे array चा उद्देश पूर्ण करणाऱ्या याद्या आहेत, परंतु त्यांची प्रक्रिया करणे स्लो आहे.
2. NumPy चे उद्देश एक अर्रे ऑब्जेक्ट (array object) तयार करणे आहे, जे Python लिस्ट पेक्षा 50x जलद आहे.
3. ही कमांड वापरून NumPy install करा:

C:\Users\Your Name>pip install numpy

3.3.1 नमपायच्या मेथड्स (Methods in numpy)

Table 3.1 Methods in numpy

all()	any()	take()	put()
apply_along_axis()	apply_over_axes()	argmin()	argmax()
nanargmin()	nanargmax()	amax()	amin()
insert()	delete()	append()	around()
flip()	fliplr()	flipud()	triu()
tril()	tri()	empty()	empty_like()
zeros()	zeros_like()	ones()	ones_like()

full_like()	diag_indices()	asmatrix()	bmat()
diagflat()	diag()	eye()	roll()
identity()	arange()	place()	extract()
compress()	rot90()	tile()	reshape()
ravel()	isinf()	isrealobj()	isscalar()
dot()	vdot()	trunc()	divide()
floor_divide()	true_divide()	random.rand()	random.randn()
ndarray.flat()	expm1()	bincount()	rint()
equal()	not_equal()	less()	less_equal()
greater()	greater_equal()	cbrt()	logical_or()
prod()	logical_not()	logical_and()	array_equal()
square()	logical_xor()	array_equiv()	sin()
cosh()	sinh()	tan()	cos()
tanh()	arcsin()	arccos()	arctan()
arctan2()			

3.3.2 अरे तयार करणे आणि फंक्शन्स initialize करणे (Creating arrays and initializing functions)

Numpy मधील अरे अनेक प्रकारे तयार केले जाऊ शकतात, विविध क्रमांकासह, अरेचा आकार परिभाषित करतात. विविध डेटा प्रकार जसे की लिस्ट, ट्युपल्स इ. वापरून अरे देखील तयार केले जाऊ शकतात. परिणामी अरेचा प्रकार अनुक्रमांमधील घटकांच्या प्रकारावरून काढला जातो.

टीप: अरे तयार करताना अरेचा प्रकार स्पष्टपणे परिभाषित केला जाऊ शकतो.

Python program for Creation of Arrays

```
import numpy as np
```

```
# Creating a rank 1 Array
```

```
arr = np.array([1, 2, 3])
```

```
print("Array with Rank 1: \n",arr)
```

```
# Creating a rank 2 Array
```

```
arr = np.array([[1, 2, 3],
                [4, 5, 6]])
```

```
print("Array with Rank 2: \n", arr)
```

```
# Creating an array from tuple
```



```
arr = np.array((1, 3, 2))
print("\nArray created using "
      "passed tuple:\n", arr)
```

Output:

Array with Rank 1:

```
[1 2 3]
```

Array with Rank 2:

```
[[1 2 3]
```

```
[4 5 6]]
```

Array created using passed tuple:

```
[1 3 2]
```

- **अरे इंडेक्समध्ये प्रवेश करणे Accessing the array Index**

numpy अरेमध्ये, अरे इंडेक्स इंडेक्स करणे किंवा ऍक्सेस करणे अनेक प्रकारे केले जाऊ शकते. अरेची श्रेणी मुद्रित करण्यासाठी, स्लाइसिंग केले जाते. अरेचे स्लाइसिंग नवीन अरेमध्ये श्रेणी परिभाषित करत आहे जी मूळ अरेमधील घटकांची श्रेणी प्रिंट करण्यासाठी वापरली जाते. कारण, स्लाइस केलेल्या अरेमध्ये मूळ अरेच्या घटकांची श्रेणी असते, कापलेल्या अरेच्या मदतीने आशय बदलल्याने मूळ अरे सामग्री सुधारते.

Python program to demonstrate

indexing in numpy array

import numpy as np

Initial Array

```
arr = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])
```

```
print("Initial Array: ")
```

```
print(arr)
```

Printing a range of Array with the use of slicing method

```
sliced_arr = arr[:2, ::2]
```

```
print ("Array with first 2 rows and"
      " alternate columns(0 and 2):\n", sliced_arr)
```

Printing elements at specific Indices

```
Index_arr = arr[[1, 1, 0, 3],
                 [3, 2, 1, 0]]
```

```
print ("\nElements at indices (1, 3), "
      "(1, 2), (0, 1), (3, 0):\n", Index_arr)
```

Output:

Initial Array:

```
[[-1.  2.  0.  4.]
```

```
[ 4. -0.5  6.  0.]
```

```
[ 2.6  0.  7.  8.]
```

```
[ 3. -7.  4.  2. ]]
```

Array with first 2 rows and alternate columns(0 and 2):

```
[[-1.  0.]
```

```
[ 4.  6.]]
```

Elements at indices (1, 3), (1, 2), (0, 1), (3, 0):

```
[0. 6. 2. 3.]
```

• मूलभूत अरे ऑपरेशन्स (Basic Array Operations):

numpy मध्ये, अरे विस्तृत ऑपरेशन्सना परवानगी देतात जी विशिष्ट अरेवर किंवा अरेच्या संयोजनावर करता येतात. या ऑपरेशनमध्ये काही मूलभूत गणितीय ऑपरेशन्स तसेच युनरी आणि बायनरी ऑपरेशन्स समाविष्ट आहेत.

Python program to demonstrate basic operations on single array

```
import numpy as np
```

```
# Defining Array 1
```

```
a = np.array([[1, 2],
               [3, 4]])
```

```
# Defining Array 2
```

```
b = np.array([[4, 3], [2, 1]])
```

```
# Adding 1 to every element
```

```
print ("Adding 1 to every element:", a + 1)
```

```
# Subtracting 2 from each element
```

```
print ("\nSubtracting 2 from each element:", b - 2)
```

```
# sum of array elements
```

```
# Performing Unary operations
```

```
print ("\nSum of all array "
```

```
    "elements: ", a.sum())
```

```
# Adding two arrays
```

```
print ("\nArray sum:\n", a + b)
```

Output:

Adding 1 to every element:

```
[[2 3]
```

```
[4 5]]
```

Subtracting 2 from each element:

```
[[ 2  1]
```

```
[ 0 -1]]
```

Sum of all array elements: 10

Array sum:

```
[[5 5]
```

```
[5 5]]
```

- **स्लाइसिंग ऑरे (Slicing arrays):**

पायथनमध्ये तुकडे करणे म्हणजे एका दिलेल्या निर्देशांकातून दुसऱ्या निर्देशांकात घटक घेणे. आम्ही इंडेक्सऐवजी स्लाइस याप्रमाणे पास करतो: [start:end]. आपण याप्रमाणे पायरी देखील परिभाषित करू शकतो: [start:end:step]. जर आम्ही पास झालो नाही तर ते 0 मानले जाते आम्ही पास न केल्यास त्या परिमाणातील अरेची मानली जाणारी लांबी संपुष्टात येईल जर आपण चरण पार केले नाही तर त्याचे 1 मानले जाते.

Example 1:

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

Output :[2,3,4,5]

Example 2

Slice elements from index 4 to the end of the array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
```

Output :[5,6,7]

Example 3

Slice elements from the beginning to index 4 (not included):

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[:4])
```

Output :[1,2,3,4]

- **उणे स्लाइसिंग (Negative Slicing):**

शेवटच्या निर्देशांकाचा संदर्भ घेण्यासाठी वजा ऑपरेटर वापरा

Example

Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])
```

Output : [5,6]

STEP Value:

Use the step value to determine the step of the slicing:

Example

Return every other element from index 1 to index 5:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
```

Output :[2,4]

Return every other element from the entire array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[::2])
```

Output :[1,3,5,7]

- **Slicing 2-D Arrays**

Example

From the second element, slice elements from index 1 to index 4 (not included):

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

Output: [7,8,,9]

- **अरेचा आकार बदलणे (Reshaping arrays)**

रिशेपिंग म्हणजे अरेचा आकार बदलणे. अरेचा आकार प्रत्येक परिमाणातील घटकांची संख्या आहे. आकार बदलून आपण परिमाण जोडू किंवा काढू शकतो किंवा प्रत्येक परिमाणात घटकांची संख्या बदलू शकतो.

1-D to 2-D आकार बदलणे (Reshape from 1-D to 2-D)

Example

Convert the following 1-D array with 12 elements into a 2-D array.

The outermost dimension will have 4 arrays, each with 3 elements:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

Output:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]
 [10 11 12]]
```

1-D to 3-D आकार बदलणे (Reshape from 1-D to 3-D)

Example

Convert the following 1-D array with 12 elements into a 3-D array.

The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(2, 3, 2)
print(newarr)
```

Output

```
[[[1 2]
```

```
[3 4]
[5 6]]
[[7 8]
[9 10]
[11 12]]]
```

• **अरे एकत्र करणे (Combining arrays)**

एकत्र होणे म्हणजे एकाच अरेमध्ये दोन किंवा अधिक अरे एकत्रित करणे. SQL मध्ये आम्ही key च्या आधारे टेबल जोडतो, तर NumPy मध्ये आम्ही एक्सिसच्या(axis) आधारे अरे जोडतो. आम्ही एक्सिस सह concatenate () फंक्शनमध्ये सामील होऊ इच्छित अरेचा एक क्रम पास करतो. अक्ष स्पष्टपणे पास केला नसल्यास, तो 0 म्हणून घेतला जातो.

Example

Join two arrays

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))
print(arr)
```

Output

```
[1 2 3 4 5 6]
```

Example

Join two 2-D arrays along rows (axis=1):

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
print(arr)
```

Output

```
[[1 2 5 6]
```

```
[3 4 7 8]]
```

• **स्टॅक फंक्शन्स वापरून अरे जोडणे (Joining Arrays Using Stack Functions):**

स्टॅकिंग हे concatenation सारखेच आहे, फरक एवढाच आहे की स्टॅकिंग नवीन अक्षावर केले जाते. आम्ही दोन 1-डी अरे दुसऱ्या अक्षावर जोडू शकतो ज्यामुळे त्यांना एक दुसऱ्यावर ठेवता येईल, म्हणजे. स्टॅकिंग stack() मेथडमध्ये एक्सिस सह जोडू इच्छित असलेल्या अरेचा क्रम आम्ही पास करतो. जर अक्ष स्पष्टपणे पास केला नसेल तर तो 0 म्हणून घेतला जातो.

Example

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
```

```
arr = np.stack((arr1, arr2), axis=1)
print(arr)
```

Output:

```
[[1 4]
 [2 5]
 [3 6]]
```

- **DataType अरेवरील गणित ऑपरेशन्स (Math Operations on DataType array)**

Numpy अरेमध्ये, अरेवर मूलभूत गणिती क्रिया घटकानुसार केल्या जातात. ही ऑपरेशन्स ऑपरेटर ओव्हरलोड्स आणि फंक्शन्स म्हणून लागू केली जातात. अरेवर गणनेसाठी अनेक उपयुक्त फंक्शन्स Numpy मध्ये दिली आहेत जसे की बेरीज: अरे घटक जोडण्यासाठी, T: घटकांच्या हस्तांतरणासाठी, इ.

Python Program to create a data type object

```
import numpy as np
# First Array
arr1 = np.array([[4, 7], [2, 6]],
                dtype = np.float64)
# Second Array
arr2 = np.array([[3, 6], [2, 8]],
                dtype = np.float64)
# Addition of two Arrays
Sum = np.add(arr1, arr2)
print("Addition of Two Arrays: ")
print(Sum)
# Addition of all Array elements using predefined sum method
Sum1 = np.sum(arr1)
print("\nAddition of Array elements: ")
print(Sum1)
# Square root of Array
Sqrt = np.sqrt(arr1)
print("\nSquare root of Array1 elements: ")
print(Sqrt)
# Transpose of Array using In-built function 'T'
Trans_arr = arr1.T
print("\nTranspose of Array: ")
print(Trans_arr)
```

Output: Addition of Two Arrays:

```
[[ 7. 13.]
 [ 4. 14.]]
```

Addition of Array elements:

```
19.0
```

Square root of Array1 elements:

```
[[2.      2.64575131]  
 [1.41421356 2.44948974]]
```

Transpose of Array:

```
[[4. 2.]  
 [7. 6.]]
```

संदर्भसूची (References) :

1. <https://www.geeksforgeeks.org/what-is-the-difference-between-pythons-module-package-and-library/>
2. <https://docs.python.org/3/tutorial/modules.html>
3. <https://realpython.com/python-modules-packages/>
4. https://www.learnpython.org/en/Modules_and_Packages

युनिट - 4

पायथन मध्ये ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (Object Oriented Programming in Python)

विषय निष्पत्ती (Course Outcome): दिलेल्या समस्येसाठी क्लास डिझाइन करा.

घटक निष्पत्ती (Theory Learning Outcome):

1. क्लास आणि ऑब्जेक्ट्स परिभाषित करण्यासाठी ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंगच्या मूलभूत संकल्पना लागू करा .
2. पॉलिमॉर्फिझमची संकल्पना लागू करा.
3. दिलेल्या समस्येसाठी इन्हेरीटन्स (वारसा) वापरून पायथन प्रोग्राम तयार करा.

4.1 ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंगची वैशिष्ट्ये (Features of object-oriented programming)

व्याख्या (Definition): ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (OOP) ही संबंधित गुणधर्म आणि वर्तन वैयक्तिक ऑब्जेक्ट्समध्ये एकत्रित करून प्रोग्रामची रचना करण्याची एक पद्धत आहे.

- ऑब्जेक्ट ओरिएंटेड म्हणजे ऑब्जेक्ट्सकडे निर्देशित.
- पायथन हे ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (OOP) आहे.
- हा प्रोग्रामिंगचा एक मार्ग आहे जो अनुप्रयोग डिझाइन आणि तयार करण्यासाठी ऑब्जेक्ट्स आणि क्लासेस वापरण्यावर लक्ष केंद्रित करतो.
- हे क्लास आणि ऑब्जेक्ट्स वापरून प्रोग्राम डिझाइन करण्यासाठी वापरले जाते.

OOP चे फायदे (Advantages of oops):

- ते जलद आहे
- कार्यान्वित करणे सोपे आहे
- हे प्रोग्रामसाठी स्पष्ट रचना प्रदान करते
- देखभाल करणे, सुधारणे आणि डीबग करणे सोपे आहे.
- कमी कोड आणि कमी डेव्हलपमेंट वेळेसह रीयुजेबिलिटी वापरता येण्याजोगे अनुप्रयोग तयार करण्यासाठी याचा वापर केला जातो

OOPची वैशिष्ट्ये (Features of oops):

- क्लास (Class)
- ऑब्जेक्ट (Object)
- एन्कॅप्सुलेशन (Encapsulation)
- अॅबस्ट्रॅक्शन (Abstraction)
- इन्हेरीटन्स (Inheritance)
- पॉलिमॉर्फिझम (Polymorphism)

1. क्लास (Class):

- क्लास हा वस्तूचा संग्रह आहे.
- हे एक तार्किक अस्तित्व आहे ज्यामध्ये काही विशिष्ट गुणधर्म आणि पद्धती आहेत.

- क्लास हा ऑब्जेक्टसाठी ब्लू प्रिंट आहे.
- क्लास हा वस्तूचा साचा आहे
- Python मधील क्लास हा डेटा आणि फंक्शन्सचा तार्किक गट आहे.
- क्लास म्हणजे वस्तूचा संग्रह

2. ऑब्जेक्ट (Object):

- ऑब्जेक्ट हे क्लास चे उदाहरण आहे.
- आक्षेपकर्त्याच्या उदाहरणामध्ये वास्तविक डेटा किंवा माहिती असते.
- वस्तू ही एक अस्तित्व आहे ज्याची स्थिती आणि वर्तन आहे.
- डेटा आणि त्या डेटावर कार्य करणाऱ्या फंक्शन्सचा संग्रह म्हणून ऑब्जेक्ट.
- एखादी वस्तू मेमरी वाटप करण्यासाठी वापरली जाते.
- प्रत्येक ऑब्जेक्टचे स्वतःचे डेटा सदस्य आणि सदस्य कार्ये असतात.

3. एन्कॅप्सुलेशन (Encapsulation) :

- डेटा आणि मेथड एकाच युनिटमध्ये गुंडाळण्याला एन्कॅप्सुलेशन म्हणतात.
- याचा वापर मेथड आणि व्हेरिएबल्समध्ये प्रवेश प्रतिबंधित करण्यासाठी केला जातो.
- एन्कॅप्सुलेशन हे उदाहरण व्हेरिएबल्स आणि दिलेल्या प्रकार (क्लास) तयार करण्याच्या पद्धती एकत्रित करण्याचे साधन आहे.
- क्लासमधील निवडक सदस्यांना त्याच्या क्लायंटकडून अगम्य ("लपलेले") बनवले जाऊ शकते, ज्याला माहिती लपवणे असे म्हणतात. माहिती लपवणे हा अमूर्ततेचा एक प्रकार आहे.

4. अ‍ॅबस्ट्रॅक्शन (Abstraction) :

- अ‍ॅबस्ट्रॅक्शनचा वापर अंतर्गत तपशील लपविण्यासाठी आणि केवळ कार्यक्षमता दाखवण्यासाठी केला जातो.
- हे पार्श्वभूमी तपशीलांचा समावेश न करता आवश्यक माहितीचा संदर्भ देते..

5. इन्हेरीटन्स (Inheritance):

- जुन्या क्लासतून नवीन क्लास काढणे याला इन्हेरीटन्स (वारसा) म्हणतात.
- जुन्या क्लासला पॅरेंट क्लास किंवा सुपर क्लास किंवा बेस क्लास म्हणतात.
- नवीन क्लासला चार्डिल्ड क्लास किंवा सबक्लास किंवा व्युत्पन्न क्लास म्हणतात.
- कोडिंगची रेऊसबिलिटी हे इन्हेरीटन्सचा मुख्य फायदा आहे.

इन्हेरीटन्स प्रकार (Types of inheritance):

1. सिंगल इन्हेरीटन्स (Single inheritance)
2. मल्टिलेवल इन्हेरीटन्स (Multilevel inheritance)
3. मल्टिपल इन्हेरीटन्स (Multiple inheritance)
4. हिरार्चिकल इन्हेरीटन्स (Hierarchical inheritance)
5. हायब्रीड इन्हेरीटन्स (Hybrid inheritance)

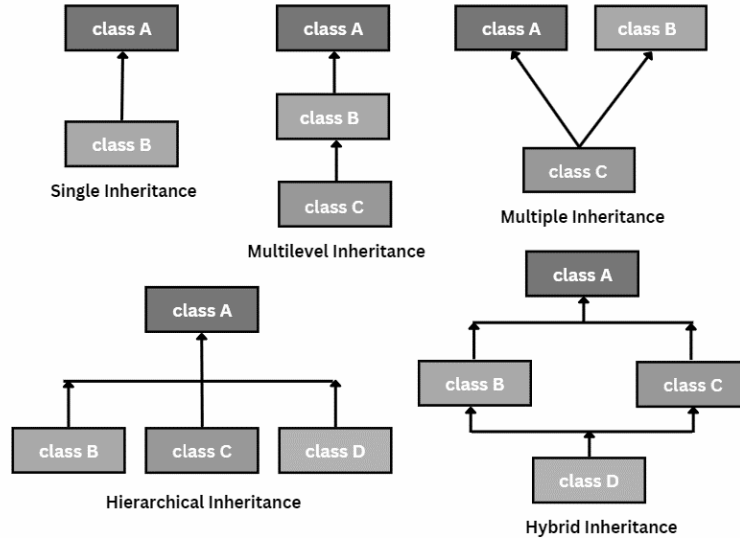


Figure 4.1: Types of Inheritance

6. पॉलिमॉर्फिझम (Polymorphism)

- पॉली म्हणजे अनेक आणि मॉर्फ म्हणजे फॉर्म.
- याचा अर्थ एकाच नावाचे एकापेक्षा जास्त फॉर्म
- याचा अर्थ एक कार्य वेगवेगळ्या प्रकारे करता येते.

पॉलिमॉर्फिझम प्रकार (Types of Polymorphism):

1. कंपाइल टाइम पॉलिमॉर्फिझम किंवा स्टॅटिक पॉलिमॉर्फिझम
 2. रन टाइम पॉलिमॉर्फिझम किंवा डायनॅमिक पॉलिमॉर्फिझम
- संकलित वेळेस वापरल्या जाणाऱ्या मेथडला कंपाइल टाइम पॉलिमॉर्फिझम म्हणतात. उदा. मेथड ओव्हरलोडिंग
 - रनटाइमच्या वेळी वापरल्या जाणाऱ्या मेथडला रन टाइम पॉलिमॉर्फिझम म्हणतात. उदा. मेथड ओव्हरराइमडिंग

4.1.1 क्लास आणि ऑब्जेक्ट तयार करणे (Creating classes and Object):

- **क्लास व्याख्या:** क्लास म्हणजे व्हेरिएबल्स आणि मेथडचा संग्रह. क्लास म्हणजे ऑब्जेक्ट चा समूह ज्यामध्ये सामान्य गुणधर्म असतात.
- पायथन क्लास हा ऑब्जेक्टचा ब्लूप्रिंट आहे.
- क्लास class हा कीवर्ड आहे

Syntax:

Class classname:

Variables and functions

- ऑब्जेक्ट तयार करणे (Creating Object) :

व्याख्या (Definition): ऑब्जेक्ट म्हणजे डेटा (व्हेरिएबल्स) आणि मेथड (फंक्शन्स) यांचा संग्रह आहे जो त्या डेटावर कार्य करतो. ऑब्जेक्ट हे क्लासचे उदाहरण आहे

Syntax: Objectname=classname()

खालील वाक्यरचना वापरून क्लासमधील व्हेरिएबल आणि फंक्शनला कॉल करा:

Objectname.variablename
Objectname.functionname()

उदाहरण:	आउटपुट:
<pre>class ruff: def f1(self): print("Hello World") ob=ruff() ob.f1()</pre>	Hello World

- **Self** व्याख्या: सेल्फ पॅरामीटर हा क्लासच्या वर्तमान उदाहरणाचा संदर्भ आहे. हे क्लासतील कोणत्याही फंक्शनचे पहिले पॅरामीटर असावे. यात ऑब्जेक्टच्या उदाहरणाचा संदर्भ आहे ज्याची मेथड संबंधित आहे.

4.1.2 कन्स्ट्रक्टर आणि डिस्ट्रक्टर (Constructor and Destructor in Python)

- **कन्स्ट्रक्टर व्याख्या (Defining constructor):**
 1. जेव्हा क्लासचा ऑब्जेक्ट तयार केला जातो तेव्हा कन्स्ट्रक्टरने क्लासच्या डेटा सदस्यांना इनिशियलाइज (व्हॅल्यूज नियुक्त करणे) करणे असते.
 2. पायथन मध्ये `_init_()` मेथडने कन्स्ट्रक्टर कॉल करतात आणि जेव्हा एखादी वस्तू तयार केली जाते तेव्हा नेहमी कॉल केली जाते.
 3. इन्स्टन्स व्हेरिएबल्सची `_init_()` मेथडमध्ये आरंभ केले जातात.

Syntax:

```
def __init__(self):
    # body of the constructor
```

उदाहरण:	आउटपुट:
<pre>class ruff: def __init__(self,a,b):self.a=a self.b=bdef f(self): print(self.a,self.b)ob=ruff(10,20) ob.f()</pre>	10 20

कन्स्ट्रक्टर प्रकार : दोन प्रकारचे कन्स्ट्रक्टर आहेत.

1. डीफॉल्ट कन्स्ट्रक्टर (Default constructor)
2. पॅरामीटराइज्ड कन्स्ट्रक्टर (Parameterized constructor)

1. डीफॉल्ट कन्स्ट्रक्टर (default constructor): डीफॉल्ट कन्स्ट्रक्टर हा साधा कन्स्ट्रक्टर आहे . कोणताही पॅरामीटर नसलेला कन्स्ट्रक्टर (No argument constructor) डीफॉल्ट कन्स्ट्रक्टर म्हणून ओळखला जातो.

उदाहरण: class ruff: def __init__(self):print("Hello") ob=ruff()	आउटपुट: Hello
---	-------------------------

2. परामीटराइज्ड कन्स्ट्रक्टर (Parameterized constructor): पॅरामीटर्स असलेल्या कन्स्ट्रक्टरला पॅरामीटराइज्ड कन्स्ट्रक्टर म्हणून ओळखले जाते. पहिला पॅरामीटर self आहे आणि उर्वरित पॅरामीटर प्रोग्रामरद्वारे प्रदान केला जातो.

उदाहरण: class ruff: def __init__(self,a,b):self.a=a self.b=b print(self.a,self.b) ob=ruff(10,20)	आउटपुट: 10 20
---	-----------------------------

• डिस्ट्रक्टर (Destructor in Python)

जेव्हा एखादा ऑब्जेक्ट नष्ट होते तेव्हा त्याला डिस्ट्रक्टर म्हणतात. पायथॉनमध्ये, c++ प्रमाणे डिस्ट्रक्टरची आवश्यकता नसते कारण पायथॉनमध्ये ऑब्जेक्ट नको असलेल्या संग्राहक असतो जो स्मृती व्यवस्थापन

Syntax:

```
del objectname
del objectname.variablename
```

स्वयंचलितपणे हाताळतो.

__del__() मेथड ही Python मध्ये डिस्ट्रक्टर पद्धत म्हणून ओळखली जाते. जेव्हा ऑब्जेक्टचे सर्व संदर्भ हटवले जातात म्हणजे जेव्हा एखादा ऑब्जेक्ट नष्ट केली जाते तेव्हा त्याला म्हणतात.

- del कीवर्ड ऑब्जेक्ट हटवण्यासाठी वापरला जातो.
- del कीवर्ड वापरून ऑब्जेक्ट्सवरील गुणधर्म हटवा.

उदाहरण: डिस्ट्रक्टरचे वर्णन करण्यासाठी पायथन प्रोग्राम

```
class Employee:
    # Initializing
    def __init__(self):
        print('Employee created.')
    # Deleting (Calling destructor)
    def __del__(self):
        print('Destructor called, Employee deleted.')
```

```
obj = Employee()
del obj
```

आउटपुट: Employee created.

Destructor called, Employee deleted.

4.2 डेटा अब्स्ट्रॅक्शन आणि डेटा एन्कॅप्सुलेशन (Data abstraction and data encapsulation):

डेटा अब्स्ट्रॅक्शन आणि डेटा एन्कॅप्सुलेशन हे ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग आणि पायथनमध्ये काम करण्याचे दोन महत्वपूर्ण सिद्धांत आहेत.

4.2.1 डेटा अब्स्ट्रॅक्शन (Data Abstraction):

डेटा अब्स्ट्रॅक्शन हा सिद्धांत डेटा विशेषता आणि वर्तन करण्याच्या उच्च स्तरावर उच्चाकांक्षा प्रोग्रामिंग मेथड आहे. ते वापरकर्त्यांना डेटा संदर्भ असताना विविध विवरणांचा नकाशा प्रदान करते, परंतु त्याचा अंदाज आणि प्रयोग कसा असेल ते दर्शवत नाही. एक उत्कृष्ट उदाहरण आहे एक कार्यालयाचे फाइल कॅबिनेट ज्यामध्ये फाइल्स असतात. प्रत्येक फाइल पात्र संदर्भासह एक नाव असते, परंतु त्याच्याकडून काही विशेष तत्वे प्रकट नसतात. या प्रकारे, डेटा अब्स्ट्रॅक्शन हे वापरकर्त्यांना डेटा तयार करण्याच्या क्रियांच्या अंदाज आणि प्रयोगाच्या अंदाजाच्या आणि निर्विघ्न तरीके प्रदान करणारे आहे.

डेटा अब्स्ट्रॅक्शन उदाहरण (Example of data abstraction):

डेटा अब्स्ट्रॅक्शन जटिल अंमलबजावणी तपशील लपवण्यावर आणि ऑब्जेक्टची केवळ आवश्यक वैशिष्ट्ये दर्शविण्यावर लक्ष केंद्रित करते. बँक खात्याचे प्रतिनिधित्व करणाऱ्या 'BankAccount' नावाच्या वर्गाचा विचार करू. आम्ही खाते क्रमांक, शिल्लक व्यवस्थापन इ. सारखे अंतर्गत तपशील उघड न करता खात्याशी संवाद साधण्यासाठी 'deposit', 'withdraw', आणि 'get_balance' सारख्या मेथड तयार करू शकतो.

```
class BankAccount:
```

```
    def __init__(self, initial_balance=0):
```

```
        self.balance = initial_balance
```

```
    def deposit(self, amount):
```

```
        self.balance += amount
```

```
        print(f"Deposited {amount} successfully.")
```

```
    def withdraw(self, amount):
```

```
        if self.balance >= amount:
```

```
            self.balance -= amount
```

```
            print(f"Withdrew {amount} successfully.")
```

```
        else:
```

```
            print("Insufficient funds.")
```

```
    def get_balance(self):
```

```
        return self.balance
```

```
# Creating an instance of BankAccount
```

```
account = BankAccount(1000)
```

```
# Depositing and withdrawing funds
```

```
account.deposit(500)
```

```

        account.withdraw(200)
# Getting the current balance
        print("Current balance:", account.get_balance())

```

या उदाहरणात, आम्ही 'deposit', 'withdraw', आणि 'get_balance' यासारख्या मेथड वापरून 'BankAccount' वर्गाशी संवाद साधतो. तथापि, बँक खात्याचे अंतर्गत तपशील, जसे की शिल्लक व्यवस्थापन यंत्रणा, वापरकर्त्यापासून दूर ठेवली जाते.

4.2.2 डेटा एन्कॅप्सुलेशन (Data Encapsulation):

डेटा एन्कॅप्सुलेशन हा सिद्धांत डेटा अस्तित्वाच्या नियंत्रणासह असताना प्रोग्रामिंग कोड कशी उत्पन्न करायला हवे याचे बाबींसाठी आहे. ते क्लासेस आणि त्यांच्या अंदरच्या विशिष्ट तत्वांच्या मदतीने संपादित केले जाते, ज्यामध्ये क्लासच्या बाह्य भागांवर प्रवेश देण्याची अनुमती नाही. एक माइक्रोवेव ओव्हन हा एक उत्कृष्ट उदाहरण आहे. आपण केवळ त्याच्या बाह्य भागाचा उपयोग करू शकता, परंतु आपल्याला केवळ त्याच्या अंदरच्या तत्वांच्या माध्यमातून त्याच्या क्रियांच्या अंदाज अथवा व्यवस्थापनाचा काम करायला हवे असेल.

एन्कॅप्सुलेशन हे उदाहरण व्हेरिएबल्स आणि दिलेल्या प्रकार (वर्ग) तयार करण्याच्या पद्धती एकत्र गुंडाळणे आहे.

- वर्गातील निवडक सदस्यांना त्याच्या क्लायंटकडून अगम्य ("लपवलेले") केले जाऊ शकते, ज्याला माहिती लपवणे असे म्हणतात.
- माहिती लपवणे हे अमूर्ततेचे स्वरूप आहे.
- वर्गातील खाजगी सदस्य दोन अंडरस्कोर वर्णानी सुरू होतात आणि थेट प्रवेश करता येत नाहीत.
- डेटा एन्कॅप्सुलेशन ऑब्जेक्टच्या काही भागांमध्ये प्रवेश प्रतिबंधित करते आणि त्याच्या अंतर्गत स्थितीत थेट बदल प्रतिबंधित करते. आम्ही खाजगी गुणधर्म वापरून आणि ऑब्जेक्टच्या स्थितीशी संवाद साधण्यासाठी सार्वजनिक मेथड प्रदान करून हे साध्य करतो.

डेटा एन्कॅप्सुलेशन उदाहरण: या उदाहरणात ' balance ' विशेषता अंतर्भूत करण्यासाठी 'BankAccount' वर्गात बदल करूया:

```

class BankAccount:
    def __init__(self, initial_balance=0):
        self.__balance = initial_balance
    def deposit(self, amount):
        self.__balance += amount
        print(f'Deposited {amount} successfully.')
    def withdraw(self, amount):
        if self.__balance >= amount:
            self.__balance -= amount
            print(f'Withdrew {amount} successfully.')
        else:
            print("Insufficient funds.")

```

```

def get_balance(self):
    return self.__balance

# Creating an instance of BankAccount
account = BankAccount(1000)

# Trying to access balance directly (will raise an error)
# print(account.__balance)

# Depositing and withdrawing funds
account.deposit(500)
account.withdraw(200)

# Getting the current balance
print("Current balance:", account.get_balance())

```

या सुधारित उदाहरणामध्ये, आम्ही 'बॅलन्स' विशेषता दुहेरी अंडरस्कोअर ('__') सह प्रीफिक्स करून खाजगी केली आहे. आता, वर्गाच्या बाहेरून थेट '__balance' मध्ये प्रवेश करण्याचा प्रयत्न केल्यास डेटा एन्कप्सुलेशन लागू करून त्रुटी निर्माण होईल. शिल्लक अॅक्सेस आता सार्वजनिक मेथड 'deposit', 'withdraw', आणि 'get_balance' द्वारे नियंत्रित केला जातो.

उदाहरण:	आउटपुट:
<pre> class ruff: def __init__(self,x,y): self.__a=x self.__b=y print(self.__a,self.__b) ob=ruff(30,20) print(ob.__a) </pre>	<pre> 30 20 AttributeError: 'ruff' object has no attribute '__a' </pre>

वरील उदाहरणात a आणि b खाजगी व्हेरिएबल्स आहेत आणि थेट प्रवेश करता येत नाहीत.

• आयडेंटिफायर्सचे नाव बदलणे याला नेम मॅंगलिंग म्हणतात.

पायथन मध्ये विशेष पद्धती (Special methods in Python):

पायथन मधील विशेष पद्धतींमध्ये दोन अंडरस्कोअर वर्णानी सुरू होणारी आणि समाप्त होणारी नावे आहेत आणि त्यांना स्वयंचलितपणे पायथनमध्ये कॉल केले जाते.

- `__init__()` - जेव्हा एखादी नवीन वस्तू तयार केली जाते तेव्हा ती स्वयंचलितपणे कॉल केली जाते.
- `__str__()` - जेव्हा एखादी वस्तू प्रिंट वापरून दाखवली जाते तेव्हा `str()` कॉल होते.
- `__repr__()` - जेव्हा पायथन शेलमध्ये ऑब्जेक्टचे मूल्य प्रदर्शित केले जाते तेव्हा `__repr__()` कॉल होते.

Table 4.1 Special Method in Python

Methods	Meaning
<code>a.__init__(self, args)</code>	constructor: <code>a = A(args)</code>
<code>a.__del__(self)</code>	destructor: <code>del a</code>
<code>a.__str__(self)</code>	pretty print: <code>print a, str(a)</code>
<code>a.__repr__(self)</code>	representation: <code>a = eval(repr(a))</code>

a.__add__(self, b)	a + b
a.__sub__(self, b)	a - b
a.__mul__(self, b)	a*b
a.__div__(self, b)	a/b
a.__lt__(self, b)	a < b
a.__gt__(self, b)	a > b
a.__le__(self, b)	a <= b
a.__ge__(self, b)	a >= b
a.__eq__(self, b)	a == b
a.__ne__(self, b)	a != b

4.3 पॉलिमॉर्फिज्मची संकल्पना- पद्धती ओव्हरलोडिंग आणि ओव्हरराइडिंग

(Concept of polymorphism- method overloading and overriding)

पॉलिमॉर्फिज्म हा शब्द ग्रीक भाषेतून आला आहे ज्याचा अर्थ "अनेक रूपे घेते असे काहीतरी" याचा अर्थ एकच फंक्शनचे नाव वेगवेगळ्या प्रकारांसाठी वापरले जाऊ शकते.

बहुरूपतेचे प्रकार

1. टाइम पॉलिमॉर्फिज्म किंवा स्टॅटिक पॉलिमॉर्फिज्म
2. रन टाइम पॉलिमॉर्फिज्म किंवा डायनॅमिक पॉलिमॉर्फिज्म

संकलित वेळेस वापरल्या जाणाऱ्या पद्धतीला कंपाइल टाइम पॉलिमॉर्फिज्म म्हणतात. उदा. मेथड ओव्हरलोडिंग, ऑपरेटर ओव्हरलोडिंग

रनटाइमच्या वेळी वापरल्या जाणाऱ्या पद्धतीला रन टाइम पॉलिमॉर्फिज्म म्हणतात. उदा. मेथड ओव्हरराइडिंग

4.3.1 बिल्ट इन पॉलिमॉर्फिज्म (Built in polymorphism in python)

<pre>a = 23 b = 11 c = 9.5 s1 = "Hello" s2 = "There!" print(a + b) print(b + c) print(s1 + s2) Output: 34 20.5 HelloThere!</pre>	<pre>str = 'HiThere' tup = ('Mon','Tue','wed','Thu','Fri') lst = ['Jan','Feb','Mar','Apr'] dict = {'1D':'Line','2D':'Triangle','3D':'Sphere'} print(len(str)) print(len(tup)) print(len(lst)) print(len(dict)) Output: 7 5 4 3</pre>
--	--

4.3.2 मेथड ओव्हरराइडिंग (Method Overriding)

मूल वर्गातील पद्धती ज्यांचे नाव पालक वर्गातील पद्धतींसारखेच आहे त्यांना मेथड ओव्हरराइडिंग म्हणून ओळखले जाते.

उदाहरण:	आउटपुट:
<pre> class one: def f1(self): print("Good morning") class two(one): def f1(self): print("Good afternoon") class three(one): def f1(self): print("Good evening") ob1=one() ob2=two() ob3=three() for a in (ob1,ob2,ob3): a.f1() </pre>	<pre> Good morning Good afternoon Good evening </pre>

4.3.3 ऑपरेटर ओव्हरलोडिंग (Operator overloading)

पायथनमधील ऑपरेटर ओव्हरलोडिंग ही ऑपरेटिंगच्या वर्ग (प्रकार) वर आधारित एकापेक्षा जास्त ऑपरेशन्स करण्याची एकल ऑपरेटरची क्षमता आहे.

• उदा.: कस्टम ऑब्जेक्ट्ससह + ऑपरेटर वापरण्यासाठी तुम्हाला `__add__` नावाची पद्धत परिभाषित करावी लागेल.

उदाहरण:	आउटपुट:
<pre> class one: def __init__(self,a,b): self.a=a self.b=b def __add__(self,other): a=self.a+other.a b=self.b+other.b ob3=one(a,b) return ob3 ob1=one(90,80) ob2=one(40,30) ob3=ob1+ob2 print(ob3.a) print(ob3.b) </pre>	<pre> 130 110 </pre>

मेथड ओव्हरलोडिंग (Method Overloading) :

मेथड ओव्हरलोडिंग हे एकाच नावाने परंतु भिन्न वितर्क असलेल्या अनेक पद्धती तयार करण्याचा एक मार्ग आहे. परंतु पायथन थेट ओव्हरलोडिंग पद्धतीला समर्थन देत नाही. परंतु अप्रत्यक्षपणे पद्धती ओव्हरलोडिंगचे समर्थन करा.

उदाहरण:	आउटपुट:
<pre> class over: def sum(self, a = None, b = None, c = None): s = 0 if a != None and b != None and c != None: s = a + b + c elif a != None and b != None: s = a + b else: s = a return s ob=over() print(ob.sum(1)) print(ob.sum(5, 5)) print(ob.sum(10, 2, 3)) </pre>	<pre> 1 10 15 </pre>

पायथनमध्ये मेथड ओव्हरलोडिंग आणि मेथड ओव्हरराइडिंगमधील फरक:
(Difference between Method Overloading and Method Overriding in Python):

Table 4.2: Difference between Overloading and Overriding

S.NO	मेथड ओव्हरलोडिंग Method Overloading	मेथड ओव्हरराइडिंग Method Overriding
1.	In the method overloading, methods or functions must have the same name and different signatures. ओव्हरलोडिंग पद्धतीमध्ये, मेथड किंवा फंक्शन्सना समान नाव आणि भिन्न स्वाक्षरी असणे आवश्यक आहे.	Whereas in the method overriding, methods or functions must have the same name and same signatures. तर मेथड ओव्हरराइडिंगमध्ये, मेथड किंवा फंक्शन्सना समान नाव आणि समान स्वाक्षर्या असणे आवश्यक आहे.
2.	Method overloading is a example of compile time polymorphism. मेथड ओव्हरलोडिंग हे कंपाइल टाइम पॉलिमॉर्फिझमचे उदाहरण आहे.	Whereas method overriding is a example of run time polymorphism. तर मेथड ओव्हरराइडिंग हे रन टाइम पॉलिमॉर्फिझमचे उदाहरण आहे.

3.	In the method overloading, inheritance may or may not be required. ओव्हरलोडिंग पद्धतीमध्ये, इन्हेरीटन्स आवश्यक असू शकतो किंवा नसू शकतो.	Whereas in method overriding, inheritance always required. मेथड ओव्हरराइडिंगमध्ये, इन्हेरीटन्स नेहमी आवश्यक असतो.
4.	Method overloading is performed between methods within the class. वर्गातील पद्धतीमध्ये मेथड ओव्हरलोडिंग केले जाते.	Whereas method overriding is done between parent class and child class methods. तर मेथड ओव्हरराइडिंग हे पालक वर्ग आणि चाइल्ड क्लास पद्धतीमध्ये केले जाते.
5.	It is used in order to add more to the behavior of methods. हे पद्धतींच्या वर्तनात अधिक जोडण्यासाठी वापरले जाते.	Whereas it is used in order to change the behavior of exist methods. जेव्हा ते विद्यमान मेथडचे वर्तन बदलण्यासाठी वापरले जाते.
6.	In method overloading, there is no need of more than one class. ओव्हरलोडिंग पद्धतीमध्ये, एकापेक्षा जास्त वर्गांची आवश्यकता नाही.	Whereas in method overriding, there is need of at least of two classes. मेथड ओव्हरराइडिंगमध्ये, किमान दोन वर्गांची आवश्यकता असते.

4.4 इन्हेरीटन्स आणि इन्हेरीटन्स प्रकार (Inheritance and types of inheritance)

इन्हेरीटन्स (वारसा) ही वर्गाची स्वतःच्या व्याख्येचा भाग म्हणून दुसऱ्या वर्गाच्या सदस्यांना वारसा देण्याची क्षमता आहे. इन्हेरिटिंग क्लासला सबक्लास ("डिराईव्हड क्लास" किंवा "चाईल्ड क्लास" [Child Class] देखील म्हणतात), आणि वारशाने मिळालेल्या वर्गाला सुपरक्लास ("बेस क्लास" किंवा "पॅरेण्ट क्लास" देखील) म्हणतात. क्लास पदानुक्रम खालीलप्रमाणे आहे:

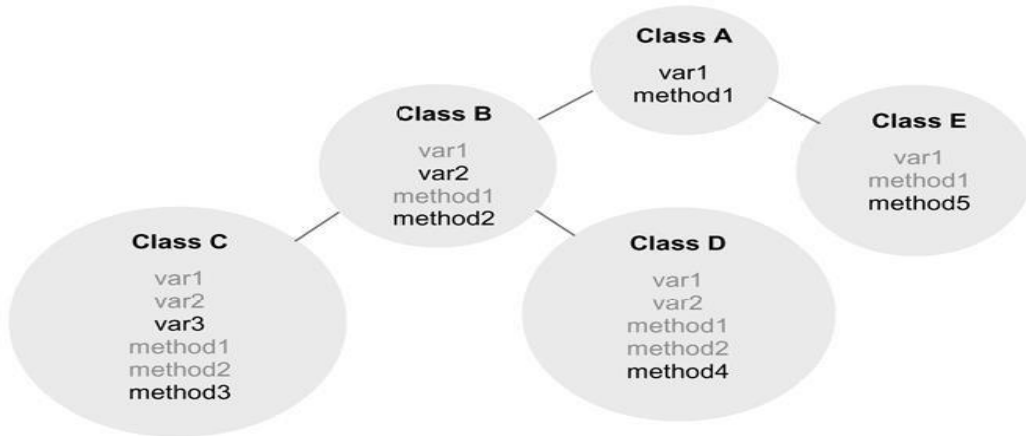


Figure 4.2:. Inheritance

वर्ग A हा सुपर क्लास आहे. क्लास B आणि E क्लास A चे सबक्लास आहेत, दोन्ही वंशपरंपरागत चल आणि क्लास A च्या पद्धती आहेत. क्लास C आणि D हे क्लास B चे थेट सबक्लास आहेत परंतु क्लास A चे अप्रत्यक्ष सबक्लास आहेत.

super() ची व्याख्या : सुपर() फंक्शन जे चाइल्ड क्लासला त्याच्या पॅरेंट क्लासकडून सर्व पद्धती आणि गुणधर्म घेतील.

इन्हेरीटन्स चे प्रकार Types of inheritance:

१. सिंगल इन्हेरीटन्स (Single inheritance)
२. मल्टिलेवल इन्हेरीटन्स (Multilevel inheritance)
३. मल्टिपल इन्हेरीटन्स (Multiple inheritance)
४. हिरार्चिकल इन्हेरीटन्स (Hierarchical inheritance)
५. हायब्रीड इन्हेरीटन्स (Hybrid inheritance)

1. सिंगल इन्हेरीटन्स (Single inheritance)

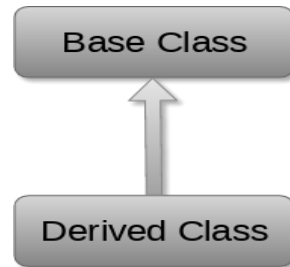


Figure 4.3: Single inheritance

सिंगल इन्हेरीटन्स मध्ये, सबक्लास फक्त एका सुपर क्लासमधून घेतला जातो. हे सिंगल -पॅरेंट क्लासचे गुणधर्म आणि वर्तन वारसा देते. कधीकधी याला साधा वारसा म्हणून देखील ओळखले जाते.

Syntax: <pre>class parent: Statementclass child(parent): Statement</pre>	
उदाहरण: <pre>class one: def f1(self): print("parent class") class two(one):def f2(self): print("child class") ob=two()ob.f1() ob.f2()</pre>	आउटपुट: <pre>parent classchild class</pre>

2. मल्टिलेवल इन्हेरीटन्स (Multilevel inheritance)

Multi-level inheritance is archived when a derived class inherits another derived class.

जेव्हा एखादा डिराईव्हड क्लास दुसऱ्या डिराईव्हड क्लासचा वारसा घेतो तेव्हा बहु-स्तरीय वारसा संग्रहित केला जातो.

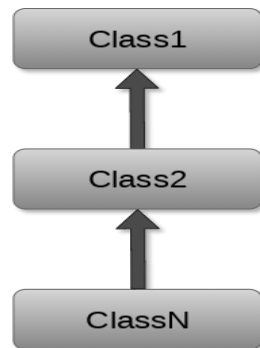


Figure 4.3: Multilevel inheritance

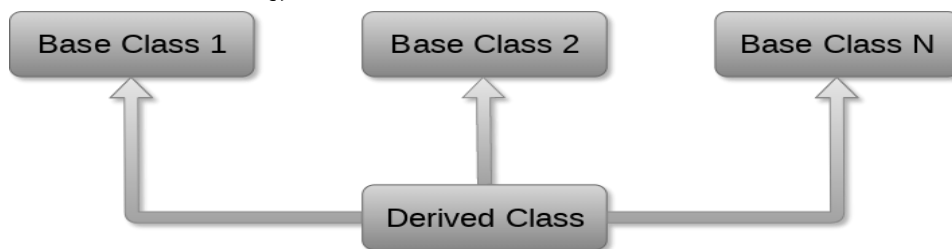
Syntax:

```

class parent:
    Statement
class child1(parent):
    Statement
class child2(child1):
    Statement
    
```

3. मल्टिपल इन्हेरीटन्स (Multiple inheritance)

एकापेक्षा जास्त पालक वर्गाकडून वारसा मिळणाऱ्या बालक वर्गाला मल्टिपल वारसा म्हणतात.



Syntax:

```

class parent 1:
    Statement
class parent 2:
    Statement
class parent n:
    Statement
class child(parent1, parent1. .... parent n):
    Statement
    
```

Figure 4.4: Multiple inheritance

उदाहरण: <pre> class one: def f1(self): print("first parent class") class two: def f2(self): print("second parent class") class three(one,two): def f3(self): print("child class") ob=three() ob.f1() ob.f2() ob.f3() </pre>	आउटपुट: <pre> first parent class second parent class child class </pre>
---	---

4. हिरार्चिकल इन्हेरीटन्स (Hierarchical inheritance) :

जर एकाच बेस क्लासमधून अनेक क्लास काढले असतील तर त्याला हिरार्चिकल वारसा म्हणतात. किंवा हा वारसा वर्गाला एकापेक्षा जास्त चाइल्ड क्लास किंवा सबक्लाससाठी पॅरेंट क्लास म्हणून होस्ट करण्याची परवानगी देतो.

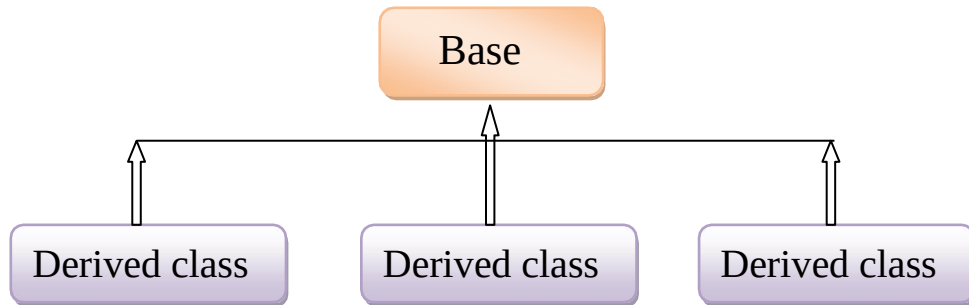


Figure 4.5: Hierarchical inheritance

Syntax:

```

class parent:
    Statement class
child1(parent):
    Statement class
child2(parent):
    Statement
class childN(parent):
    Statement

```

उदाहरण: <pre> class one: def f1(self): print("parent class") class two(one): def f2(self): print("Intermediate parent class") class three(two): def f3(self): print("child class") ob=three()ob.f1() ob.f2() ob.f3() </pre>	आउटपुट: parent class Intermediate parent class child class
---	--

6. हायब्रीड इन्हेरीटन्स (Hybrid inheritance) :

हायब्रीड म्हणजे एकापेक्षा जास्त. हायब्रीड वारसा हे दोन किंवा अधिक प्रकारच्या वारशाचे संयोजन आहे.

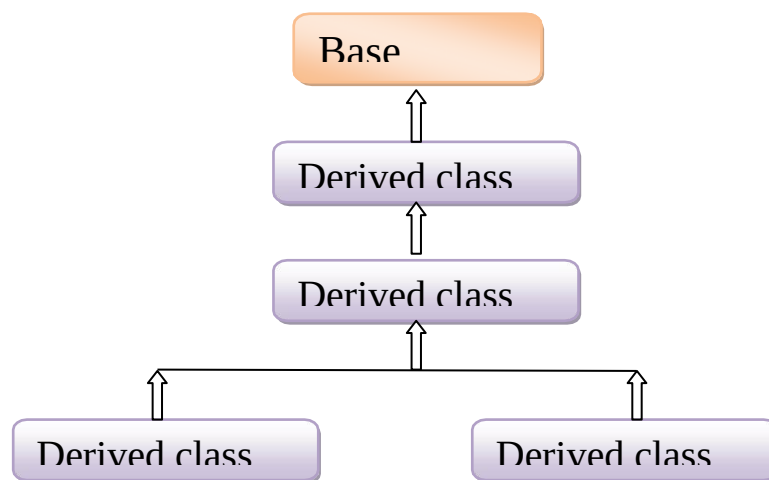


Figure 4.6: Hybrid inheritance

उदाहरण: <pre> class one: def f1(self): print("first parent class") class two: def f2(self): print("second parent class") class three(two): def f3(self): print("child class one") </pre>	आउटपुट: first parent class second parent class child class one child class two
--	--

<pre>class four(one,three): def f4(self): print("child class two") ob=four()ob.f1() ob.f2() ob.f3() ob.f4()</pre>	
---	--

संदर्भसूची (References) :

1. <https://realpython.com/python3-object-oriented-programming/>
2. <https://www.programiz.com/python-programming/object-oriented-programming>
3. <https://www.geeksforgeeks.org/python-oops-concepts/>
4. <https://www.pythontutorial.net/python-oop/>
5. https://www.youtube.com/watch?v=Ej_02ICOIgs
6. https://www.tutorialspoint.com/object_oriented_python/object_oriented_python_tutorial.pdf
7. https://www.w3schools.com/python/python_classes.asp

युनिट- 5

लिनियर डेटा स्ट्रक्चर , अरे, लिंक लिस्ट, स्टॅक आणि क्यूज युझिंग पायथन

(Linear Data Structure Arrays, Link List, Stack and Queues using Python)

विषय निष्पत्ती (Course Outcome) : लिनियर डेटा स्ट्रक्चर इन पायथन

घटक निष्पत्ती (Theory Learning Outcome):

1. पायथनमधील लिनियर डेटा संरचनाचे वर्णन करा.
2. दिलेल्या समस्येसाठी अरे वापरून पायथन कोड विकसित करा.
3. दिलेल्या समस्येसाठी लिंक लिस्ट यादी लागू करण्यासाठी पायथन कोड लिहा.
4. पायथन प्रोग्राममध्ये स्टॅक लागू करा.
5. पायथन प्रोग्राममध्ये क्यू लागू करा.

5.2 डेटा स्ट्रक्चर्स - व्याख्या, लिनियर डेटा स्ट्रक्चर्स, नॉन-लिनियर डेटा स्ट्रक्चर्स अरे - आढावा, अरेचे प्रकार, अरेवरील ऑपरेशन्स, अरे व लिस्ट यांची तुलना (Data Structures - definition, linear data structures, non-linear data structures arrays - overview, types of arrays, operations on arrays, arrays vs list)

"डेटा स्ट्रक्चर" हा संगणक विज्ञानातील एक महत्वाचा विषय आहे. तो साधारणपणे डेटा संरचना आणि मांडणी करणे कसे करावे याबद्दल शिकवतो. डेटा स्ट्रक्चरचा अभ्यास कोणत्याही प्रक्रियेची किंवा प्रोग्रामची कार्यक्षमता वाढवण्यासाठी डेटाची संघटना आणि डेटा मांडणी व्यवस्थापन समजून घेण्यास मदत होते.

डेटा स्ट्रक्चर एक प्रकारची विचारणा आहे जे डेटा रूपांतरण, संगठन, आणि प्रवाह कसे होणार याबद्दल तंत्रज्ञान माहिती आणि ते कसे प्रोग्रामिंग भाषेत लिहिण्याच्या संरचनात मदत करते. तो डेटा संग्रहित करणे, संगठित करणे, आणि प्रक्रियांसाठी वापरण्याच्या उपायांचे अभ्यास म्हणजे डेटा स्ट्रक्चर.

डेटा संरचना म्हणजे माहिती किंवा डेटा सुसंगत पद्धतीने आयोजित करण्याची एक पद्धत, ज्यामुळे त्याच्यावर प्रक्रिया करणे अधिक सोयीचे आणि कार्यक्षम होते. तो डेटाची संघटना, संगठन, आणि प्रवाह साधारणपणे विविध प्रकारे केले जातात, जेणेकरून हा डेटा सहजपणे पुनर्प्राप्त केला जाऊ शकतो आणि आवश्यकतेनुसार भविष्यात कार्यक्षमतेने वापरला जाऊ शकतो.

• विशिष्ट डेटा मॉडेलची व्याप्ती दोन घटकांवर अवलंबून असते:

प्रथम, वास्तविक-जगातील ऑब्जेक्ट्सह डेटाचा निश्चित सहसंबंध प्रतिबिंबित करण्यासाठी ते संरचनेमध्ये पुरेसे लोड केले जाणे आवश्यक आहे.

दुसरे, निर्मिती इतकी सरळ असावी की जेव्हा आवश्यक असेल तेव्हा डेटावर कार्यक्षमतेने प्रक्रिया करण्यास अनुकूल असेल .

डेटा स्ट्रक्चर्सची काही उदाहरणे म्हणजे अरे, लिंक लिस्ट, स्टॅक, क्यू, ट्री इ. डेटा स्ट्रक्चर्स संगणक विज्ञानाच्या जवळजवळ प्रत्येक पैलूमध्ये, म्हणजे, कंपाइलर डिझाइन, ऑपरेटिंग सिस्टम, ग्राफिक्स, आर्टिफिशियल इंटेलिजन्स आणि बरेच काही मध्ये मोठ्या प्रमाणावर वापरले जातात.

डेटा स्ट्रक्चर्स हे अनेक संगणक विज्ञान अल्गोरिदमचे मुख्य भाग आहेत कारण ते प्रोग्रामरना डेटा प्रभावी प्रकारे वापरण्यासाठी मदत करतात आणि कोड कोरचे साधन आहे.

• डेटा स्ट्रक्चर्स (Data Structure) :

डेटा स्ट्रक्चर्स हे कोणत्याही सॉफ्टवेअर किंवा प्रोग्रामचे बिल्डिंग ब्लॉक्स असतात. प्रोग्रामसाठी योग्य डेटा स्ट्रक्चर निवडणे हे प्रोग्रामरसाठी अत्यंत आव्हानात्मक काम आहे.

जेव्हा जेव्हा डेटा संरचनांचा समावेश होतो तेव्हा खालील काही मूलभूत संज्ञा वापरल्या जातात:

- ❖ **डेटा(Data):** डेटाला प्राथमिक मूल्य किंवा मूल्यांचा संग्रह म्हणून परिभाषित करू शकतो. उदाहरणार्थ, कर्मचार्याचे नाव आणि आयडी हा कर्मचार्याशी संबंधित डेटा आहे.
- ❖ **डेटा आयटम(Data Item):** मूल्याचे एक एकक डेटा आयटम म्हणून ओळखले जाते.
- ❖ **ग्रुप आयटम(Group Item):** डेटा आयटम असलेल्या डेटा आयटम समूह आयटम म्हणून ओळखले जातात. उदाहरणार्थ, एखाद्या कर्मचार्याच्या नावाचे नाव, मधले आणि आडनाव असू शकते.
- ❖ **प्राथमिक वस्तू(first Element):** उप-आयटममध्ये विभागण्यात आलेल्या डेटा आयटमला प्राथमिक वस्तू म्हणून ओळखले जाते. उदाहरणार्थ, कर्मचार्याचा आयडी.
- ❖ **अस्तित्व आणि गुणधर्म:** विशिष्ट वस्तूचा वर्ग एखाद्या घटकाद्वारे दर्शविला जातो. त्यात विविध गुणधर्म असतात. प्रत्येक विशेषता त्या घटकाच्या विशिष्ट गुणधर्माचे प्रतीक आहे.
- ❖ **डेटा स्ट्रक्चर्सचे वर्गीकरण (Classification of data structure)**
डेटा स्ट्रक्चर विविध प्रकारे एकमेकांशी संबंधित व्हेरिएबल्सचा संरचित संच वितरित करते. हे प्रोग्रामिंग साधनाचा आधार बनवते जे डेटा घटकांमधील संबंध दर्शवते आणि प्रोग्रामरना डेटावर कार्यक्षमतेने प्रक्रिया करण्यास मदत करते.

डेटा स्ट्रक्चर्सचे दोन श्रेणींमध्ये वर्गीकरण करू शकतो: प्रिमिटिव्ह डेटा स्ट्रक्चर आणि नॉन-प्रिमिटिव्ह डेटा स्ट्रक्चर

खालील आकृती डेटा स्ट्रक्चर्सचे वेगवेगळे वर्गीकरण दर्शवते.

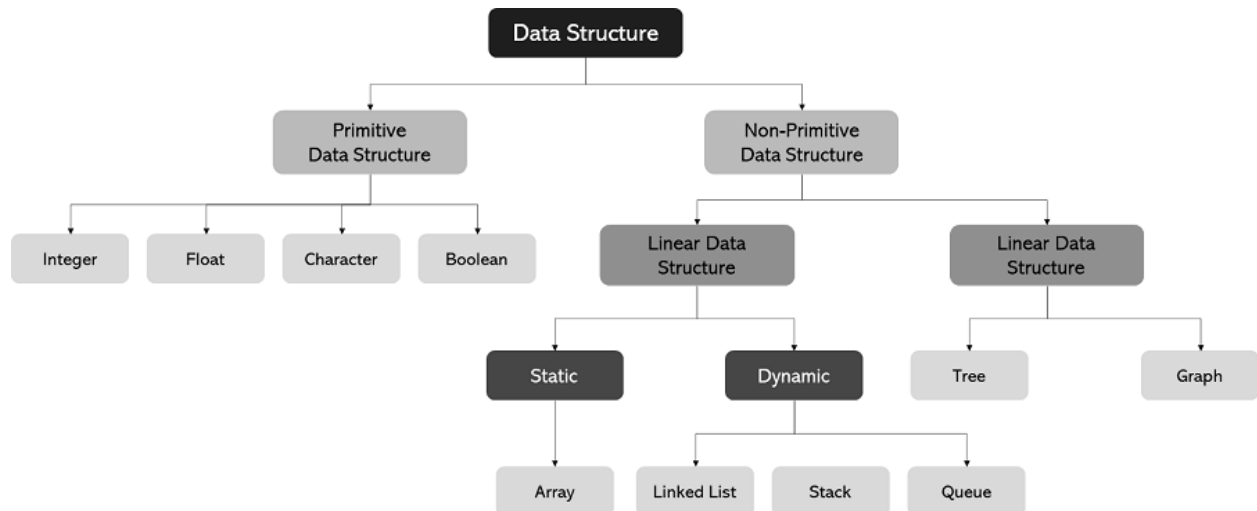


Figure 5.1 : Different types of data structure

❖ प्रिमिटिव्ह डेटा स्ट्रक्चर्स(Primitive Data Structure):

प्रिमिटिव्ह डेटा स्ट्रक्चर्स ही डेटा स्ट्रक्चर्स आहेत ज्यात संख्या आणि वर्ण असतात जे प्रोग्राममध्ये अंतर्भूत असतात. या डेटा स्ट्रक्चर्समध्ये फेरफार किंवा मशीन-स्तरीय निर्देशांद्वारे थेट ऑपरेट केले जाऊ शकते. इंटिजर, फ्लोट, कॅरेक्टर आणि बुलियन सारखे मूलभूत डेटा प्रकार आदिम डेटा स्ट्रक्चर्स अंतर्गत येतात. या डेटा प्रकारांना साधे डेटा प्रकार देखील म्हणतात, कारण त्यात वर्ण आहेत ज्यांना अधिक विभागले जाऊ शकत नाही.

❖ नॉन-प्रिमिटिव्ह डेटा स्ट्रक्चर्स(Non Primitive Data Structure):

नॉन-प्रिमिटिव्ह डेटा स्ट्रक्चर्स म्हणजे प्रिमिटिव्ह डेटा स्ट्रक्चर्समधून व्युत्पन्न केलेल्या डेटा स्ट्रक्चर्स. या डेटा स्ट्रक्चर्समध्ये फेरफार किंवा मशीन-स्तरीय सूचनांद्वारे थेट ऑपरेट केले जाऊ शकत नाही. या डेटा स्ट्रक्चर्सचा फोकस डेटा घटकांचा एक संच तयार करण्यावर आहे जो एकसंध (समान डेटा प्रकार) किंवा विषम (भिन्न डेटा प्रकार) आहे. डेटाची रचना आणि व्यवस्थेच्या आधारावर, आपण या डेटा स्ट्रक्चर्सचे दोन उप-श्रेणींमध्ये विभाजन करू शकतो -

1. लिनियर डेटा संरचना
2. नॉन-लिनियर डेटा स्ट्रक्चर्स

❖ लिनियर डेटा संरचना(Linear Data Structure):

डेटा स्ट्रक्चर जी त्याच्या डेटा घटकांमध्ये एकलिनियर कनेक्शन जतन करते त्याला लिनियर डेटा स्ट्रक्चर म्हणून ओळखले जाते. डेटाची मांडणी रेषीय पद्धतीने केली जाते, जिथे प्रत्येक घटकामध्ये पहिला आणि शेवटचा डेटा घटक वगळता उत्तराधिकारी आणि पूर्ववर्ती असतात. तथापि, मेमरीच्या बाबतीत ते खरे असेलच असे नाही, कारण व्यवस्था अनुक्रमिक असू शकत नाही.

मेमरी वाटपाच्या आधारावर, लिनियर डेटा स्ट्रक्चर्सचे पुढील दोन प्रकारांमध्ये वर्गीकरण केले जाते:

❖ स्टॅटिक डेटा स्ट्रक्चर्स(Static Data Structure):

स्थिर आकार असलेल्या डेटा स्ट्रक्चर्सला स्टॅटिक डेटा स्ट्रक्चर्स म्हणतात. या डेटा स्ट्रक्चर्ससाठी मेमरी कंपाइलरच्या वेळी वाटप केली जाते, आणि संकलित केल्यानंतर वापरकर्त्याद्वारे त्यांचा आकार बदलता येत नाही; तथापि, त्यांच्यामध्ये संग्रहित डेटा बदलला जाऊ शकतो.

अरे हे स्टॅटिक डेटा स्ट्रक्चरचे सर्वोत्तम उदाहरण आहे कारण त्यांचा आकार निश्चित आहे आणि त्याचा डेटा नंतर बदलला जाऊ शकतो.

❖ डायनॅमिक डेटा स्ट्रक्चर्स(Dynamic Data Structure):

डायनॅमिक आकार असलेल्या डेटा स्ट्रक्चर्सला डायनॅमिक डेटा स्ट्रक्चर्स म्हणतात. या डेटा स्ट्रक्चर्सची मेमरी रन टाइममध्ये वाटप केली जाते आणि कोडच्या रन टाइममध्ये त्यांचा आकार बदलतो. शिवाय, वापरकर्ता कोडच्या रन टाइममध्ये या डेटा स्ट्रक्चर्समध्ये संग्रहित केलेला आकार तसेच डेटा घटक बदलू शकतो.

लिंक लिस्ट, स्टॅक आणि क्यू ही डायनॅमिक डेटा स्ट्रक्चर्सची सामान्य उदाहरणे आहेत.

लिनियर डेटा स्ट्रक्चर्सचे प्रकार(Linear Data Structure Types):

1 . अरे(Array)

अरे ही डेटा संरचना आहे . ज्याचा वापर एकाच डेटा प्रकारातील एकाधिक डेटा घटक एका व्हेरिएबलमध्ये गोळा करण्यासाठी केला जातो. समान डेटा प्रकारांची अनेक व्हॅल्यूज वेगळ्या व्हेरिएबल नावांमध्ये संग्रहित करण्याऐवजी, ती सर्व एकाच व्हेरिएबलमध्ये संग्रहित करू शकतो. या विधानाचा अर्थ असा नाही की आपल्याला कोणत्याही प्रोग्राममधील समान डेटा प्रकाराची सर्व मूल्ये त्या डेटा प्रकाराच्या एका अर्रेमध्ये एकत्र करावी लागतील. परंतु असे काही वेळा असतील जेव्हा समान डेटा प्रकारांचे काही विशिष्ट व्हेरिएबल्स अर्रेसाठी योग्य प्रकारे एकमेकांशी संबंधित असतात.

अरे ही घटकांची एक सूची आहे जिथे प्रत्येक घटकाला सूचीमध्ये एक वेगळे स्थान असते. अर्रेचे डेटा घटक समान व्हेरिएबल नाव शेअर करतात; तथापि, प्रत्येकास सबस्क्रिप्ट म्हणतात भिन्न अनुक्रमणिका क्रमांक असतो. सूचीतील कोणत्याही डेटा एलिमेंटमध्ये त्याच्या स्थानाच्या मदतीने प्रवेश करू शकतो. अशा प्रकारे, समजून घेण्यासाठी अर्रेचे मुख्य वैशिष्ट्य म्हणजे डेटा संलग्न मेमरी स्थानांमध्ये संग्रहित केला जातो, ज्यामुळे वापरकर्त्यांना त्यांच्या संबंधित अनुक्रमणिका वापरून अर्रेच्या डेटा घटकांमधून मार्गक्रमण करणे शक्य होते.

उदाहरण :- कारांची नावे असलेली एक अरे तयार करा

```
cars = ["Ford", "Volvo", "BMW"]
```

❖ अक्सेस अर्रेच्या इलीमेंट्स (Access Array Elements):

आपण अनुक्रमांकाचा संदर्भ घेऊन अरे घटकाचा संदर्भ देऊ शकता.

उदाहरण

पहिल्या अरे घटकाची किंमत मिळवा:

```
x = cars[0]
```

पहिल्या अरे इलीमेंटस किंमत बदला:

```
cars[0] = "Toyota"
```

❖ अर्रेची लांबी(length):

अर्रेची लांबी (अर्रेमधील घटकांची संख्या) परत करण्यासाठी len() पद्धत वापरा.

उदाहरण

cars अर्रेमधील घटकांची संख्या परत करा:

```
x = len(cars)
```

अर्रेच्या घटकांवर लूप करणे

तुम्ही for in लूप वापरून अर्रेमधील सर्व घटकांवर लूप करू शकता.

उदाहरण

cars अर्रेमधील प्रत्येक घटक मुद्रित करा:

```
for x in cars:
```

```
    print(x)
```

❖ अर्रेमध्ये घटक जोडणे(Adding the elements in array):

तुम्ही append() पद्धत वापरून अर्रेमध्ये घटक जोडू शकता.

उदाहरण

`cars` ऐरमध्ये आणखी एक घटक जोडा:

```
cars.append("Honda")
```

❖ **ऐरमधील घटक काढून टाकणे(Poping the array element):**

तुम्ही `pop()` पद्धत वापरून ऐरमधून घटक काढू शकता.

उदाहरण : `cars` ऐरमधील दुसरा घटक काढून टाका

```
cars.pop(1)
```

तुम्ही `remove()` पद्धत वापरून देखील ऐरमधून घटक काढू शकता.

उदाहरण

"Volvo" मूल्य असलेला घटक काढून टाका:

```
cars.remove("Volvo")
```

Python मध्ये, `array` आणि `list` या दोन वेगवेगळ्या डेटा स्ट्रक्चर आहेत, आणि त्यांच्यात काही महत्वाचे फरक आहेत. खालील मजकूरात आपण या फरकांबद्दल चर्चा करू.

लिस्ट (List):**1 . हेटरोजिनियस (heterogeneous):**

- ❖ Python लिस्ट ही हेटरोजिनियस (heterogeneous) असू शकते, म्हणजेच एका लिस्टमध्ये विविध प्रकारचे डेटा (`int`, `float`, `string`, इ.) असू शकतात.

- ❖ उदाहरण:

```
my_list = [1, 2.5, "hello", True]
```

2 . डायनामिक साईज(Dynamic Size):

- ❖ लिस्टचे आकार डायनामिक असतात, म्हणजेच त्यात आवश्यकतेनुसार आयटम्स ऍड किंवा रिमूव करता येतात.

- ❖ उदाहरण:

```
my_list.append(3) # लिस्टमध्ये नवीन आयटम ऍड करणे
```

```
my_list.remove(2.5) # लिस्टमधून आयटम रिमूव करणे
```

3 . बिल्ट-इन सपोर्ट(Built In Support):

- ❖ लिस्ट हे Python च्या बिल्ट-इन डेटा प्रकारांपैकी एक आहे, म्हणून कोणतेही अतिरिक्त मॉड्यूल इम्पोर्ट करण्याची गरज नाही.

- ❖ उदाहरण:

```
my_list = [1, 2, 3]
```

अरे (Array):**1. होमोजिनियस (Homogeneous) :**

- ❖ अरे मध्ये सर्व आयटम्स एकाच प्रकारचे असणे आवश्यक आहे (उदा. सगळे इंटिजर, सगळे फ्लोट्स इ.).

- ❖ उदाहरण:

```
from array import array
```

```
my_array = array('i', [1, 2, 3, 4])
```

2. फिक्स्ड टाइप(Fixed Type):

- ❖ अरे मध्ये डेटा टाइप निर्दिष्ट करावा लागतो. उदाहरणार्थ, 'i' म्हणजे integer, 'f' म्हणजे float.
- ❖ उदाहरण:

```
my_array = array('i', [1, 2, 3, 4]) # 'i' indicates integers
```

अतिरिक्त मॉड्यूलची गरज(Need of Extra Modules):

- ❖ अरे वापरण्यासाठी Python च्या `array` मॉड्यूलची गरज असते.
- ❖ उदाहरण:

```
from array import array
```

लिस्ट व अरेची तुलना (Comparison of List and Array):

वैशिष्ट्य	लिस्ट (List)	अरे (Array)
डेटा प्रकार	हेटरोजिनियस (heterogeneous) जिथे एकाच डेटास्ट्रक्चरमध्ये विविध प्रकारचे डेटा असू शकतात.	होमोजिनियस (homogeneous) जिथे एकाच प्रकारचा डेटा एकत्रित केला जातो.
साईज	डायनामिक(Dynamic)	स्टॅटिक(static)
इम्पोर्ट	बिल्ट-इन	`array` मॉड्यूल इम्पोर्ट आवश्यक
लवचिकता	जास्त	कमी
पफॉर्मन्स	मोठ्या डेटा सेटसाठी कमी कार्यक्षम	मोठ्या डेटा सेटसाठी जास्त कार्यक्षम

केव्हा कोणता वापरावा?

लिस्ट : जेव्हा आपल्याला विविध प्रकारचा डेटा ठेवायचा असतो किंवा डायनामिक साईज आवश्यक असतो.

अरे : जेव्हा आपल्याला एकाच प्रकारचा मोठा डेटा सेट ठेवायचा असतो आणि अधिक कार्यक्षमतेची गरज असते.

या फरकांमुळे, Python मध्ये कोणता डेटा स्ट्रक्चर कधी वापरायचा हे ठरवणे सोपे होते. आपल्याला विशेष गरजांनुसार लिस्ट किंवा अरे निवडता येतील.

5.2 सर्चिंग / शोधणे -लिनिअर सर्च आणि बायनरी सर्च , सोर्ट / क्रमवारी- बबल सोर्ट , इन्सार्पण

सोर्ट (Searching -linear search and binary search, sorting bubble sort, insertion sort):

Linear Search (लिनिअर सर्च):

लिनिअर सर्च हे एक सरळ सर्च आहे ज्यातून आपण एका सरणीतील प्रत्येक एलेमेंटसाठी प्रत्येक घटकाची तुलना करता जाते आणि सर्चता जाते. या सर्चाचा समयग्रहण $O(n)$ असतो, जेथे 'n' सरणीच्या एलेमेंटची संख्या आहे.

उदाहरणार्थ, Python मध्ये लिनिअर सर्चचा कोड:

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i # Return index if found
    return -1 # Return -1 if not found
# उदाहरण (Example)
arr = [5, 3, 8, 4, 2]
target = 8
result = linear_search(arr, target)
if result != -1:
    print("Current state: {result}")
else:
    print("Element not found")
```

OutPut:

Current state : 2

1. Binary Search (बायनरी सर्च):

बायनरी सर्च हे सामान सर्चण्याचा प्रभावी आणि लघुचालक आणि विभाजनाच्या व्याख्यानावर आधारित आहे. आपल्याला शोधण्यासाठी एलेमेंटची सामान्यता नाही, बरळविच्छेदण्याच्या प्रक्रियेत घटक असतात. सामान चालीत करताना, आपण मध्ये सारखे घटक चालीत करता जातो आणि एक संख्येवर शोधलेली संख्या समान असलेल्या प्रत्येक घटकाच्या विरुद्ध मार्ग चालीत करता जातो.

उदाहरणार्थ, Python मध्ये बायनरी चा कोड:

```
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid # Return index if found
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1 # Return -1 if not found
```

उदाहरण

```
arr = [2, 4, 6, 8, 10, 12, 14, 16]
target = 10
result = binary_search(arr, target)
if result != -1:
    print("Current state: {result}")
```

```
else:
    print("Element not found")
```

OutPut:

Current state: 4

1. Bubble Sort (बबल सॉर्ट):

बबल सॉर्ट हे सरळ बटणारी सॉर्ट आहे ज्यामध्ये प्रत्येक घटकाला बदलून स्थानांतरीत केली जाते, त्यामुळे नवीनतम संख्या संचार करते. संचारणाची प्रक्रिया हे सर्वच घटकांच्या सामान्यतेत घटक सज्जा करते. बबल सॉर्टचा समयग्रहण $O(n^2)$ आहे, ज्यामध्ये 'n' संख्यांची संख्या आहे.

उदाहरणार्थ, Python मध्ये बबल सॉर्टचा कोड:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

उदाहरण

```
arr = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(arr)
print("Sorted:")
print(arr)
```

OutPut:

```
Sorted :
[11, 12, 22, 25, 34, 64, 90]
```

2. Insertion Sort (इन्सर्शन सॉर्ट):

इन्सर्शन सॉर्ट हे विचारप्रद आणि सर्वसाधारण सॉर्ट आहे ज्यामध्ये प्रत्येक घटक सरळ्याने सामान सापडतो. इन्सर्शन सॉर्टचा समयग्रहण $O(n^2)$ आहे, ज्यामध्ये 'n' संख्यांची संख्या आहे.

उदाहरणार्थ, Python मध्ये इन्सर्शन सॉर्टचा कोड:

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```

उदाहरण

```
arr = [12, 11, 13, 5, 6]
insertion_sort(arr)
```



```
print(" Sorted:")
```

```
print(arr)
```

OutPut:

Sorted :

[5, 6, 11, 12, 13]

5.3 लिंक लिस्ट - सिंगली लिंक लिस्ट, डबली लिंक लिस्ट, सर्क्युलर लिंक लिस्ट, पायथन पॅकेजेस

वापरून लिंक लिस्ट (Linked Lists - singly linked list, doubly linked list, circular linked lists, implementation using Python packages for link list):

लिंक लिस्ट(Linked list):

लिंक लिस्ट ही एक डेटा स्ट्रक्चर आहे ज्याचा डेटा संघटन आणि व्यवस्थापनात महत्त्वाचा रोल आहे. यात काही नोड्सचा समावेश आहे जे मेमरीमध्ये रँडम ठिकाणी साठवलेले असतात, ज्यामुळे मेमरी व्यवस्थापन प्रभावी होते. प्रत्येक नोडमध्ये दोन मुख्य घटक असतात: डेटा भाग आणि पुढील नोडचा संदर्भ.

लिंक लिस्ट का? (Why link List)

लिंक लिस्टची निर्मिती सामान्य लिस्ट्स आणि ऍरेजमध्ये डेटा साठवण्याशी संबंधित विविध त्रुटींचा निराकरण करण्यासाठी करण्यात आली आहे, जसे की:

इन्सेरशन आणि डिलेशन ऑफ एलिमेंट (Insertion and deletion of Element)

लिस्ट्समध्ये, शेवटी सोडून इतर कोणत्याही स्थानी एक घटक घालणे किंवा काढणे आवश्यक असते तेव्हा सर्व नंतरच्या वस्तूंना वेगवेगळ्या स्थानी शिफ्ट करणे आवश्यक असते. हा प्रक्रिया वेळ $O(n)$ घ्यायचा आणि लिस्टचा आकार वाढल्यास कामगिरी लक्षणीयरीत्या घटू शकते. जर तुम्हाला आधीच कसे लिस्ट्स कार्य करतात किंवा त्यांची अंमलबजावणी माहित नसेल, तर पायथन लिस्ट्सवरील ट्युटोरियल वाचू शकता.

लिंक लिस्ट्स मात्र वेगळ्या प्रकारे काम करतात. त्या विविध, सलग नसलेल्या मेमरी ठिकाणांमध्ये घटक साठवतात आणि पुढील नोड्सशी पॉइंटर्सच्या माध्यमातून त्यांना जोडतात. ही रचना लिंक लिस्ट्सना कोणत्याही स्थानी घटक घालणे किंवा काढणे सुलभ करते, फक्त लिंक्स बदलून नवीन घटक जोडणे किंवा हटवलेले वगळणे शक्य असते.

एकदा घटकाचा स्थान ओळखल्यानंतर आणि घालणे किंवा काढण्याचे स्थान थेट प्रवेशात असल्यास, नोड्स जोडणे किंवा हटवणे $O(1)$ वेळेत केले जाऊ शकते.

डायनॅमिक साईज (Dynamic Size)

पायथन लिस्ट्स डायनॅमिक ऍरेज आहेत, ज्याचा अर्थ त्यांना आकार बदलण्याची लवचिकता आहे. तथापि, या प्रक्रियेत अनेक जटिल कार्ये समाविष्ट आहेत, ज्यात ऍरेला नवीन, मोठ्या मेमरी ब्लॉकमध्ये पुन्हा वाटप करणे समाविष्ट आहे. असे पुन्हा वाटप कार्यक्षम नसते कारण घटक नवीन ब्लॉकमध्ये कॉपी केले जातात, संभाव्यतः आवश्यकतेपेक्षा अधिक जागा वाटप करतात. विपरीत, लिंक लिस्ट्स पुन्हा वाटप किंवा रीझायझिंगची आवश्यकता नसताना डायनॅमिकरीत्या वाढू किंवा कमी होऊ शकतात. त्यामुळे ज्या कार्यासाठी उच्च लवचिकता आवश्यक आहे, त्यांच्यासाठी त्या एक आदर्श पर्याय बनतात.

मेमरी कार्यक्षमता (Memory Operation)

लिस्ट्स त्यांच्या सर्व घटकांसाठी एक सलग ब्लॉक मेमरी वाटप करतात. जर एक लिस्ट त्याच्या आरंभीच्या आकारापेक्षा जास्त वाढवायची असेल, तर तिला एक नवीन, मोठा सलग मेमरी ब्लॉक वाटप करावा लागतो

आणि नंतर सर्व विद्यमान घटक त्या नवीन ब्लॉकमध्ये कॉपी करावे लागतात. हा प्रक्रिया वेळखाऊ आणि कार्यक्षम नसते, विशेषतः मोठ्या लिस्ट्ससाठी. दुसरीकडे, जर लिस्टचा आरंभीचा आकार जास्त ठरवला गेला असेल, तर वापरात नसलेली मेमरी वाया जाते. विपरीत, लिंक लिस्ट्स प्रत्येक घटकासाठी स्वतंत्रपणे मेमरी वाटप करतात. ही रचना नवीन घटक जोडल्यास मेमरी चांगल्या प्रकारे वापरली जाते.

कधी लिंक लिस्ट्स वापराव्यात? (When to use Linked List?)

लिंक लिस्ट्स सामान्य लिस्ट्स आणि ऍरेजपेक्षा काही फायदे देतात, जसे की डायनॅमिक आकार आणि मेमरी कार्यक्षमता, परंतु त्यांच्याकडे काही मर्यादा देखील आहेत. प्रत्येक घटकासाठी पुढील नोड संदर्भासाठी पॉइंटर्स साठवावे लागतात, त्यामुळे लिंक लिस्ट्सचा मेमरी वापर प्रति घटक जास्त असतो. तसेच, या डेटा स्ट्रक्चरमुळे थेट डेटा प्रवेश शक्य होत नाही. एखादा घटक शोधण्यासाठी सुरुवातीपासून लिस्टमधून श्रेणीसुधारित करणे आवश्यक असते, ज्यामुळे $O(n)$ शोध वेळ लागतो. लिंक लिस्ट किंवा ऍरे यापैकी कोणते निवडावे हे ॲप्लिकेशनच्या विशिष्ट गरजांवर अवलंबून असते. लिंक लिस्ट्स विशेषतः उपयुक्त असतात जेव्हा:

- ❖ तुम्हाला वारंवार अनेक घटक घालावे किंवा हटवावे लागतात
- ❖ डेटा आकार अनपेक्षित किंवा वारंवार बदलणारा असतो
- ❖ घटकांमध्ये थेट प्रवेशाची आवश्यकता नसते
- ❖ डेटासेटमध्ये मोठे घटक किंवा रचना असतात

लिंक लिस्ट्सचे प्रकार (Types of Linked List)

तीन प्रकारच्या लिंक लिस्ट्स आहेत, प्रत्येक विविध परिस्थितीसाठी अद्वितीय फायदे देतात. हे प्रकार आहेत:

1.सिंगली-लिंक लिस्ट्स(Singly Linked List):

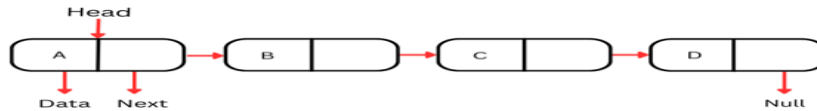


Figure 5.2 : Singly Linked List

सिंगली-लिंक लिस्ट हा सर्वात सोपा लिंक लिस्ट प्रकार आहे, ज्यात प्रत्येक नोडमध्ये काही डेटा आणि पुढील नोडचा संदर्भ असतो. त्या केवळ एकाच दिशेने प्रवास करता येतात - हेड (पहिला नोड) पासून टेल (शेवटचा नोड) पर्यंत.

प्रत्येक नोडमध्ये दोन भाग असतात:

- ❖ **डेटा (Data):** नोडमध्ये साठवलेली वास्तविक माहिती.
- ❖ **नेक्स्ट पॉइंटर(Next Pointer):** पुढील नोडचा संदर्भ. शेवटच्या नोडचा नेक्स्ट पॉइंटर सहसा null ला सेट केला जातो.

हे डेटा स्ट्रक्चर केवळ एकाच दिशेने प्रवास करू शकतात, त्यामुळे विशिष्ट मूल्य किंवा इंडेक्सद्वारे एखादा घटक शोधण्यासाठी हेडपासून सुरू करून नोड्समधून श्रेणीसुधारित करणे आवश्यक असते. या ऑपरेशनला $O(n)$ वेळ लागतो, ज्यामुळे मोठ्या लिस्ट्ससाठी कमी कार्यक्षम होते.

सिंगली-लिंक लिस्टच्या सुरुवातीला नोड घालणे आणि हटवणे अत्यंत कार्यक्षम आहे, ज्याची वेळ $O(1)$ असते. तथापि, मध्यम किंवा शेवटी घालणे आणि हटवणे आवश्यक स्थानी पर्यंत श्रेणीसुधारित करणे आवश्यक असते, ज्यामुळे $O(n)$ वेळ लागतो.

सिंगली-लिंक लिस्ट्सची रचना त्या कार्यासाठी उपयुक्त डेटा स्ट्रक्चर बनवते जेव्हा ऑपरेशन्स लिस्टच्या सुरुवातीला होतात.

2. डबलि-लिंक लिस्ट्स(doubly linked list):

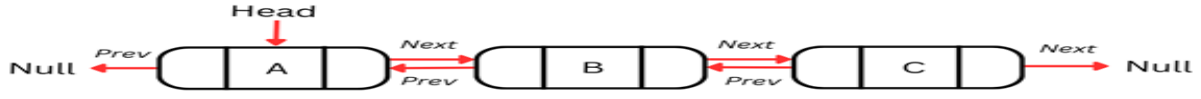


Figure 5.3 : Doubly Linked List

सिंगली-लिंक लिस्ट्सचे एक तोटा म्हणजे आपण त्यांना फक्त एकाच दिशेने प्रवास करू शकतो आणि आवश्यकता असल्यास मागील नोडकडे पुन्हा प्रवास करू शकत नाही. ही मर्यादा आपल्याला द्विदिशात्मक नेव्हिगेशनची आवश्यकता असलेल्या ऑपरेशन्स करण्याची क्षमता कमी करते.

डबलि-लिंक लिस्ट्स हा समस्या सोडवतो एका अतिरिक्त पॉइंटरच्या समावेशाने, ज्यामुळे लिस्ट दोन्ही दिशेने प्रवास करता येते. प्रत्येक नोडमध्ये तीन घटक असतात: डेटा, पुढील नोडचा पॉइंटर आणि मागील नोडचा पॉइंटर.

3. सर्क्युलर लिंक लिस्ट्स(Circular Linked List):

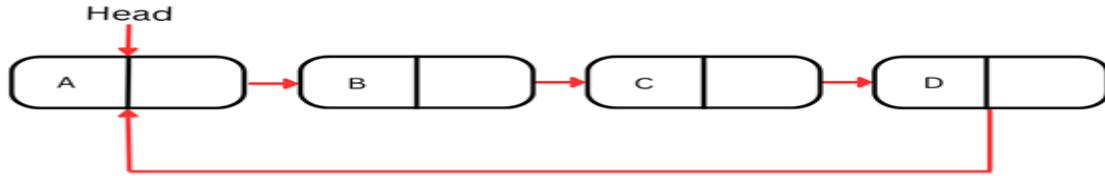


Figure 5.4: Circular Linked List

सर्क्युलर लिंक लिस्ट्स हे एक विशेष प्रकारच्या लिंक लिस्ट्स आहेत ज्यात शेवटचा नोड पहिल्या नोडकडे पुन्हा पोहोचतो, एक सर्क्युलर संरचना तयार करतो. याचा अर्थ सिंगली आणि डबलि लिंक लिस्ट्सप्रमाणे, सर्क्युलर लिंक लिस्ट संपत नाही; ती लूप होते.

सर्क्युलर लिंक लिस्ट्सचे सायक्लिकल स्वरूप त्या सतत लूप कराव्या लागणाऱ्या परिस्थितीसाठी आदर्श बनवते, जसे की बोर्ड गेम्स जे शेवटच्या खेळाडूपासून पहिल्या खेळाडूपर्यंत पुन्हा सुरू होतात, किंवा राउंड-रॉबिन शेड्यूलिंग सारख्या संगणकीय अल्गोरिदममध्ये.

• पायथन मध्ये लिंक लिस्ट तयार करणे (Creating linked list in Python)

या LinkedList class मध्ये, आपण Node class चा वापर करून linked list तयार करू. या class मध्ये, आपल्याकडे एक `__init__` method आहे जे linked list ला एक रिकाम्या head ने initialize करते. त्यानंतर, आमच्याकडे `insertAtBegin()` method आहे ज्याने linked list च्या सुरुवातीला एक node insert करतो, `insertAtIndex()` method दिलेल्या index वर एक node insert करतो, आणि `insertAtEnd()` method linked list च्या शेवटी एक node insert करतो. त्यानंतर, आमच्याकडे `remove_node()` method आहे जो data ला argument म्हणून घेतो आणि त्या node ला delete करतो. `remove_node()` method मध्ये

आम्ही linked list traverse करतो, जर data समान असलेला node उपस्थित असेल तर आम्ही त्या node ला linked list मधून delete करतो. मग आमच्याकडे sizeofLL() method आहे जो linked list चा सध्याचा size मिळवतो आणि LinkedList class चा शेवटचा method आहे printLL() जो linked list traverse करतो आणि प्रत्येक node चा data print करतो.

1. Node Class तयार करणे

आपण एक Node class तयार केली आहे ज्यामध्ये आपण __init__ function परिभाषित केला आहे जो node ला argument म्हणून पास केलेल्या data ने initialize करतो आणि एक reference None ने सेट करतो कारण जर आपल्याकडे फक्त एक node असेल तर त्याच्या reference मध्ये काहीही नसते.

class Node:

```
def __init__(self, data):
    self.data = data
    self.next = None
```

2. लिंक लिस्ट मध्ये Insertion

लिंक लिस्ट च्या सुरुवातीला Insertion

हा method linked list च्या सुरुवातीला node insert करतो. या method मध्ये, आपण दिलेल्या data सह एक new_node तयार करतो आणि तपासतो की head एक रिकामा node आहे की नाही. जर head रिकामा असेल तर आपण new_node ला head बनवतो आणि return करतो, अन्यथा आपण head ला new_node च्या next मध्ये insert करतो आणि head ला new_node च्या बरोबर सेट करतो.

```
def insertAtBegin(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
        return
    else:
        new_node.next = self.head
        self.head = new_node
```

3. लिंक लिस्ट मध्ये विशिष्ट स्थानावर Node Insert करणे

हा method दिलेल्या index वर linked list मध्ये node insert करतो. या method मध्ये, आपण दिलेल्या data सह एक new_node तयार करतो, एक current_node जो head च्या बरोबर असतो, आणि एक counter 'position' ज्याची सुरुवात 0 ने होते. आता, जर index शून्य असेल तर याचा अर्थ node सुरुवातीला insert करायचा आहे, त्यामुळे आपण insertAtBegin() method कॉल करतो, अन्यथा आपण एक while loop चालवतो जोपर्यंत current_node None च्या बरोबर नाही किंवा (position+1) दिलेल्या index च्या बरोबर नाही. आपल्याला दिलेल्या स्थानावर insert करण्यासाठी एक स्थान मागे राहावे लागते, ज्यामुळे nodes चा linking होईल, आणि प्रत्येक iteration मध्ये, आपण position 1 ने वाढवतो आणि current_node त्याच्या next च्या बरोबर करतो. जेव्हा loop संपते आणि जर current_node None च्या बरोबर नाही तर आपण current_node नंतर new_node insert करतो. जर current_node None च्या बरोबर असेल तर याचा अर्थ index list मध्ये अस्तित्वात नाही आणि आपण "Index not present" असे print करतो.

Indexing starts from 0.

```
def insertAtIndex(self, data, index):
    new_node = Node(data)
    current_node = self.head
    position = 0
    if position == index:
        self.insertAtBegin(data)
    else:
        while(current_node != None and position+1 != index):
            position = position+1
            current_node = current_node.next

        if current_node != None:
            new_node.next = current_node.next
            current_node.next = new_node
        else:
            print("Index not present")
```

4. लिंक लिस्ट च्या शेवटी Insertion

हा method linked list च्या शेवटी node insert करतो. या method मध्ये, आपण दिलेल्या data सह एक new_node तयार करतो आणि तपासतो की head एक रिकामा node आहे की नाही. जर head रिकामा असेल तर आपण new_node ला head बनवतो आणि return करतो. अन्यथा, आपण current_node ला head च्या बरोबर करतो आणि linked list च्या शेवटच्या node पर्यंत traverse करतो. जेव्हा current_node नंतर None मिळते तेव्हा while loop संपते आणि आपण current_node च्या next मध्ये जो linked list चा शेवटचा node आहे तिथे new_node insert करतो.

```
def insertAtEnd(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
        return
    current_node = self.head
    while(current_node.next):
        current_node = current_node.next
    current_node.next = new_node
```

5. लिंक लिस्ट चा Node अपडेट करणे

हे code linked list class मध्ये updateNode नावाचा method परिभाषित करतो. हा method दिलेल्या स्थानावर linked list मधील node चे मूल्य अपडेट करण्यासाठी वापरला जातो.

```
# Update node of a linked list
# at given position
```

```
def updateNode(self, val, index):
    current_node = self.head
    position = 0
    if position == index:
        current_node.data = val
    else:
        while(current_node != None and position != index):
            position = position+1
            current_node = current_node.next
        if current_node != None:
            current_node.data = val
        else:
            print("Index not present")
```

6. लिंक लिस्ट मधून Node काढणे

Linked List मधून पहिला Node काढणे

हा method linked list चा पहिला node काढतो, फक्त दुसऱ्या node ला linked list चा head बनवून.

```
def remove_first_node(self):
    if(self.head == None):
        return
    self.head = self.head.next
```

7. दिलेल्या स्थानावर Linked List मधील Node काढणे

या method मध्ये, आपण दिलेल्या index वरचा node काढू, हा method insert_at_index() method सारखाच आहे. या method मध्ये, जर head None असेल तर आपण फक्त return करतो, अन्यथा आपण self.head सह एक current_node आणि 0 सह position initialize करतो. जर position दिलेल्या index च्या बरोबर असेल तर आपण remove_first_node() method कॉल करतो, अन्यथा आपण एक node काढण्यापूर्वी एका node पर्यंत जाण्यासाठी while loop वापरून traverse करतो. त्यानंतर, जेव्हा आपण while loop मधून बाहेर येतो तेव्हा आपण तपासतो की current_node None च्या बरोबर आहे का, जर नाही तर आपण current_node च्या next ला आपण काढू इच्छित node च्या next च्या बरोबर करतो, अन्यथा आपण "Index not present" असा संदेश print करतो कारण current_node None च्या बरोबर आहे.

Method to remove at given index

```
def remove_at_index(self, index):
    if self.head == None:
        return
    current_node = self.head
    position = 0
    if position == index:
        self.remove_first_node()
```

```

else:
    while(current_node != None and position+1 != index):
        position = position+1
        current_node = current_node.next
    if current_node != None:
        current_node.next = current_node.next.next
    else:
        print("Index not present")

```

8. दिलेल्या डेटाने Linked List मधील Node काढणे

हा method दिलेल्या डेटाने linked list मधील node काढतो. या method मध्ये, पहिल्यांदा आपण current_node च्या head बरोबर करतो आणि linked list च्या traverse करण्यासाठी while loop चालवतो. हे while loop त्यामुळे बंद होते की current_node None होतो किंवा current node च्या next च्या डेटा next ते argument मधील डेटाशी समान असते. आता, while loop मधून बाहेर येताना जर current_node None असेल तर त्याचा अर्थ आहे की त्या डेटा वर्गीकरण डेटा मधील node अस्तित्वात नाही आणि आपण फक्त return करतो, आणि जर current_node च्या next च्या डेटा next दिलेल्या डेटाशी समान असेल तर आपण त्या node ला काढून current_node च्या next च्या डेटा next बनवतो. आणि हे if else condition वापरून केले जाते.

```

def remove_node(self, data):
    current_node = self.head
    # Check if the head node contains the specified data
    if current_node.data == data:
        self.remove_first_node()
        return
    while current_node is not None and current_node.next.data != data:
        current_node = current_node.next
    if current_node is None:
        return
    else:
        current_node.next = current_node.next.next

```

9. पायथन मध्ये लिंक लिस्ट चे भ्रमण Traversal

हा method linked list चे traverse करतो आणि प्रत्येक node चा data print करतो. या method मध्ये, आपण current_node च्या head बरोबर करतो आणि while loop वापरून linked list च्या traverse करतो तोता current_node None होतो किंवा current_node च्या data हर iteration मध्ये print करतो आणि current_node च्या next ला करतो.

```

def printLL(self):
    current_node = self.head
    while(current_node):
        print(current_node.data)

```

```
current_node = current_node.next
```

❖ पायथन मध्ये लिंक लिस्ट चा उदाहरण

या उदाहरणात, Node आणि LinkedList class वर परिभाषित करण्यानंतर, आम्ही LinkedList class वापरून "l1" नावाचा एक linked list तयार केला आणि नंतर त्यामध्ये character data 'a', 'b', 'c', 'd' आणि 'g' सहित चार nodes insert केले. नंतर आम्ही printLL() method वापरून linked list चा प्रिंट केला. नंतर, आम्ही काही nodes काढून remove methods वापरून काढले आणि नंतर पुन्हा linked list चा प्रिंट केला आणि पाहू शकतो की node सफळतेने काढले गेले आहे. नंतर, आम्ही linked list चा साइज print केला आहे.

```
# Node class to define each node in the linked list
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

# LinkedList class to manage the linked list operations
class LinkedList:
    def __init__(self):
        self.head = None

    # Method to insert a node at the beginning of the linked list
    def insertAtBegin(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    # Method to print the linked list
    def printLL(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

    # Method to remove a node with given data
    def removeNode(self, data):
        current = self.head
        if current and current.data == data:
            self.head = current.next
            return
        prev = None
        while current and current.data != data:
            prev = current
            current = current.next
```



```

    if current is None:
        print("Node with data", data, "not found.")
        return
    prev.next = current.next
# Method to get the size of the linked list
def getSize(self):
    current = self.head
    size = 0
    while current:
        size += 1
        current = current.next
    return size
# Example usage of the LinkedList class
if __name__ == "__main__":
    llist = LinkedList()
    llist.insertAtBegin('d')
    llist.insertAtBegin('c')
    llist.insertAtBegin('b')
    llist.insertAtBegin('a')
    llist.insertAtBegin('g') # Adding 'g' at the beginning
    print("Initial Linked List:")
    llist.printLL()
    # Removing nodes
    llist.removeNode('g') # Remove 'g'
    llist.removeNode('d') # Remove 'd'
    print("\nLinked List after removals:")
    llist.printLL()
    # Printing size of the linked list
    print("\nSize of the Linked List:", llist.getSize())
या कोडचा प्रिंट हा अंदाज देतो:
Initial Linked List:
a -> b -> c -> d -> g -> None
Linked List after removals:
a -> b -> c -> None
Size of the Linked List: 3
यामध्ये, आपण linked list मध्ये 'a', 'b', 'c', 'd' आणि 'g' या character data सहित चार nodes insert
केलेल्या आणि त्यानंतर 'g' आणि 'd' nodes काढून remove केलेल्या आहेत.
# Node class for a single node in singly linked list
class Node:
    def __init__(self, data):

```

```
        self.data = data
        self.next = None
# Linked List class to manage operations
class SinglyLinkedList:
    def __init__(self):
        self.head = None
    def print_list(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")
# Example usage
if __name__ == "__main__":
    sll = SinglyLinkedList()
    sll.head = Node(1)
    second = Node(2)
    third = Node(3)
    sll.head.next = second
    second.next = third
    sll.print_list()
# Node class for a single node in doubly linked list
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None
# Doubly Linked List class to manage operations
class DoublyLinkedList:
    def __init__(self):
        self.head = None
    def print_list(self):
        current = self.head
        while current:
            print(current.data, end=" <-> ")
            current = current.next
        print("None")
# Example usage
if __name__ == "__main__":
    dll = DoublyLinkedList()
```

```
dll.head = Node(1)
second = Node(2)
third = Node(3)
dll.head.next = second
second.prev = dll.head
second.next = third
third.prev = second
dll.print_list()
# Node class for a single node in circular linked list
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
# Circular Linked List class to manage operations
class CircularLinkedList:
    def __init__(self):
        self.head = None
    def print_list(self):
        if self.head is None:
            print("List is empty")
            return
        current = self.head
        while True:
            print(current.data, end=" -> ")
            current = current.next
            if current == self.head:
                break
        print()
    def insert_at_end(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            new_node.next = self.head
        else:
            current = self.head
            while current.next != self.head:
                current = current.next
            current.next = new_node
            new_node.next = self.head
# Example usage
```

```
if __name__ == "__main__":
    cll = CircularLinkedList()
    cll.insert_at_end(1)
    cll.insert_at_end(2)
    cll.insert_at_end(3)
    cll.print_list()
```

OutPut:

1 -> 2 -> 3 ->

5.4. स्टॅक: स्टॅकचा परिचय, स्टॅक ॲप्लिकेशन्स - एक्सप्रेशन इव्हलुएशन , बॅकट्रॅकिंग, ट्रॅव्हर्सल - इनफिक्स, प्रिफिक्स आणि पोस्टफिक्स संकल्पना(Stacks: introduction to stacks, stack applications - expression evaluation, backtracking, traversal - infix, prefix and postfix concepts):

स्टॅक (Stack):

स्टॅक हे एक डेटा स्ट्रक्चर आहे जे शेवटचा आला तो प्रथम (Last In, First Out - LIFO) या तत्वाचे पालन करते, जिथे सर्वात अलीकडे जोडलेला घटक पहिला काढला जातो. Python मध्ये, स्टॅक विविध पद्धतींनी अंमलात आणता येतो, जसे की लिस्ट वापरून किंवा collections.deque मॉड्यूलचा वापर करून.

- **लिस्ट वापरून (Using List)**

पायथन मध्ये लिस्टचा वापर करून स्टॅक तयार केला जाऊ शकतो. **append()** आणि **pop()** पद्धतींनी हे करता येते.

पायथन चा लिस्ट डेटा स्ट्रक्चर स्टॅक अंमलबजावणीसाठी वापरला जातो. यामध्ये **append()** आणि **pop()** मेथड्सचा वापर करून स्टॅक व्यवस्थापन केले जाते.

उदाहरण:**लिस्ट वापरण्याची उदाहरण**

```
stack = []
```

- ❖ **घटक घालणे (Insertion of element)**

```
stack.append('a')
stack.append('b')
stack.append('c')
print("Stack:", stack)
```

- ❖ **घटक काढणे (Removing of element)**

```
print("remove element:", stack.pop())
print("after removal Stack:", stack)
```

- **collections.deque वापरणे**

पायथन मध्ये **collections.deque** हा एक प्रकारचा लाइनियर डेटा संरचना आहे जो लिस्टपेक्षा खूप कार्यक्षम असतो जिथे डेटा एका टोकावरून किंवा दुसऱ्या टोकावरून काढता (**pop**) किंवा टाकता (**append**) येतो. **deque** वापरून तुम्ही एकूणच पॉप, अपेंड इत्यादी कार्ये **O(1)** वेळेत (**constant time**) करू शकता .

उदाहरण:

```
from collections import deque
# deque वापरण्यांची उदाहरण
stack = deque()
# घटक घालणे
stack.append('a')
stack.append('b')
stack.append('c')
print("Stack:", stack)
# घटक काढणे
print("remove element:", stack.pop())
print("After removal:", stack)
```

- **queue.LifoQueue वापरणे**

उदाहरण:

```
from queue import LifoQueue
# LifoQueue वापरण्यांची उदाहरण
stack = LifoQueue()
# घटक घालणे
stack.put('a')
stack.put('b')
stack.put('c')
print("Stack:", stack.queue)
# घटक काढणे
print("remove element:", stack.get())
print("after removal:", stack.queue)
```

स्टॅक ऑपरेशन्स(Stack Operations):

- **एम्प्टी स्टॅक(empty stack):** स्टॅक रिक्त आहे का हे ओळखण्यासाठी `len(stack) == 0` किंवा `stack.empty()` (LifoQueue साठी) वापरले जाते.

```
if not stack:
    print("empty")
else:
    print("not empty")
```
- **साइज(size):** स्टॅकमध्ये असलेल्या घटकांची संख्या मिळवण्यासाठी `len(stack)` वापरले जाते.

```
print(len(stack))
```
- **टोप इलीमेंट(top element):** ह्या ऑपरेशनसाठी `stack[-1]` (लिस्टसाठी) आणि `stack[-1]` (deque साठी) वापरले जाते. LifoQueue साठी, `stack.queue[-1]` वापरा.

```
if stack:
    top_element = stack[-1]
```

```
print("Top element:", top_element)
else:
    print("empty")
```

- **पुश(push) :** स्टॅकमध्ये घटक जोडण्यासाठी `append()` (लिस्टसाठी) आणि `append()` (deque साठी) वापरले जाते. LifoQueue साठी, `put()` वापरा.

```
stack.append('d')
print("After push:", stack)
```

- **पॉप(pop) :** स्टॅकमध्ये शीर्ष घटक काढण्यासाठी `pop()` (लिस्टसाठी) आणि `pop()` (deque साठी) वापरले जाते. LifoQueue साठी, `get()` वापरा.

```
top_element = stack.pop()
print("Popped element :", top_element)
print("After Pop:", stack)
```

स्टॅकच्या फायदे(Benefits of Stack):

- ❖ स्टॅक हे सोप्या डेटा संरचना आहेत ज्यांना स्पष्टपणे परिभाषित केलेल्या कार्यक्रमांची संचालन करण्यात यथार्थ आणि सुलभ बनवते.
- ❖ स्टॅकमध्ये घटके जोडण्याचे आणि काढण्याचे प्रक्रिया वेळेच्या संकल्पनेत $O(1)$ समयचर्या असतात, ज्यामुळे हे क्रियांचे कार्य प्रभावीरित्या करते.
- ❖ स्टॅक डेटा संरचनेमध्ये घटकांच्या क्रमाची वळण करण्यासाठी वापरले जाते.

स्टॅकची अयोग्यता(Disadvantages of stack):

- ❖ स्टॅकमध्ये साइज एकदा ठरली कि त्या पूर्ण झाल्यास, आपण स्टॅकमध्ये आणखी कोणतेही घटक जोडू शकत नाही.
- ❖ स्टॅक वर तपासणी करण्याची वेगवान योग्यता नाही, कारण आपण वर तपासण्याच्या जास्तीत जास्त घटके काढवून त्यांच्या घटकात तपासून शोधणे गरजेचे असते.

स्टॅक (Stack) ऍप्लिकेशन्समध्ये एक उपयुक्त वापर आहे एक्सप्रेशन इव्हॅल्यूएशन (Expression Evaluation) साठी. आपण Python वापरून एक्सप्रेशन इव्हॅल्यूएशन कसे करता येईल, याबद्दल खालील उदाहरणांमध्ये मराठीत समजावू.

एक्सप्रेशन इव्हॅल्यूएशन उदाहरण (Exempl of Expresssion evolution)

जर, आपल्याला एक्सप्रेशन ``3 + (4 * 5)`` विचारले आहे आणि त्याचा मूल्य काढण्याचा आवश्यकता आहे.

```
def evaluate_expression(expression):
```

```
    stack = []
```

```
    operators = {'+': lambda x, y: x + y,
```

```
                '-': lambda x, y: x - y,
```

```
                '*': lambda x, y: x * y,
```

```
                '/': lambda x, y: x / y}
```

```
    i = 0
```

```
    while i < len(expression):
```

```
        if expression[i].isdigit():
```

```

num = 0
while i < len(expression) and expression[i].isdigit():
    num = num * 10 + int(expression[i])
    i += 1
stack.append(num)
elif expression[i] in operators or expression[i] == '(':
    stack.append(expression[i])
    i += 1
elif expression[i] == ')':
    num2 = stack.pop()
    op = stack.pop()
    num1 = stack.pop()
    if op in operators:
        result = operators[op](num1, num2)
        stack.append(result)
    i += 1
else:
    i += 1
while len(stack) > 1:
    num2 = stack.pop()
    op = stack.pop()
    num1 = stack.pop()
    if op in operators:
        result = operators[op](num1, num2)
        stack.append(result)
return stack[0]

```

उदाहरण एक्सप्रेसन

```

expression = "3 + (4 * 5)"
print(f"Expression: {expression}")
print(f"Result: {evaluate_expression(expression)}")

```

Output :

Expression: 3 + (4 * 5)

Result: 23

एक्सप्रेसनचे मूल्य काढा (Calculation of expression value)

```

result = evaluate_expression(expression)
print(f"एक्सप्रेसन {expression} चा मूल्य: {result}")

```

या उदाहरणात, `evaluate\_expression` नावाच्या फंक्शनचा वापर केला जातो ज्यात स्टॅकचा वापर करून एक्सप्रेसन सोडविल्या जातात. एक्सप्रेसन मध्ये अंक आणि ऑपरेटर्सचा संचयन करून, आपल्याला एक्सप्रेसनचे मूल्य परत मिळतो.

या प्रकारे, स्टॅक ऍप्लिकेशन्समध्ये एक्सप्लेन इव्हल्यूएशन Python मध्ये सोप्या पद्धतीने केले जाते.

- ❖ स्टॅक (Stack) ऍप्लिकेशन्समध्ये बॅकट्रॅकिंग (Backtracking) हा एक महत्वाचा अल्गोरिदम आहे ज्याचा उपयोग केला जातो विविध समस्यांच्या वर्तमानातून निदर्शन करण्यासाठी. या अल्गोरिदमचा उपयोग करून आपण समस्यांच्या सर्व संभाव्य समाधाने प्राप्त करू शकता.

बॅकट्रॅकिंग उदाहरण (Example of Backtracking)

सर्व संभाव्य दारात विचारलेल्या मार्गाचा उदाहरण घेऊया. आपल्याला दिलेल्या मार्गावर सर्व अंधारी संभाव्य वाहतूके करण्यात येतील.

```
def backtrack(path, choices):
    if len(path) == len(choices):
        print(path) # छापा प्राप्त करा, समस्येचे एक संभाव्य समाधान
        return
    for choice in choices[len(path)]:
        if choice not in path:
            path.append(choice)
            backtrack(path, choices)
            path.pop()
```

उदाहरण: मार्गावर वाहतूक करणारी अंधारी संभाव्यता

```
choices = [
    ['Status', 'Choice A', 'Status', 'Status'],
    ['Status', 'Status', 'Status', 'Status'],
    ['Choice A', 'Choice A', 'Choice A', 'Status'],
    ['Status', 'Status', 'Status', 'Darkness'],
]
print("Backtracking Solutions:") backtrack([], choices)
```

Output :

Backtracking Solutions:

```
['Status', 'Choice A', 'Status', 'Status']
['Status', 'Choice A', 'Status', 'Darkness']
['Status', 'Status', 'Status', 'Darkness']
['Choice A', 'Status', 'Status', 'Darkness']
```

❖ स्टॅक ऍप्लिकेशन्स - ट्रॅव्हर्सल - इनफिक्स, प्रिफिक्स आणि पोस्टफिक्स (Stack Applications- Traversal-Infix, prefix, postfix)

स्टॅक ऍप्लिकेशन्समध्ये ट्रॅव्हर्सल म्हणजेच विचरणाचे तीन प्रमुख प्रकार आहेत: इनफिक्स, प्रिफिक्स, आणि पोस्टफिक्स. या प्रकारांना आम्ही पायथॉन कोडच्या माध्यमातून समजून घेऊ.

1. इनफिक्स (Infix):

इनफिक्स नोंदणीमध्ये ऑपरेटर त्यांच्या दोन ऑपरेन्डच्या मध्ये असतात. उदाहरण: `A + B`.

2. प्रिफिक्स (Prefix):

प्रिफिक्स नोंदणीमध्ये ऑपरेटर त्यांच्या दोन ऑपरेन्डच्या आधी येतो. उदाहरण: `+ A B`.

3. पोस्टफिक्स (Postfix):

पोस्टफिक्स नोंदणीमध्ये ऑपरेटर त्यांच्या दोन ऑपरेन्डच्या नंतर येतो. उदाहरण: `A B +`.

स्टॅक क्लास: प्रथम, आपण एक साधी स्टॅक क्लास तयार करूया.

```
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return len(self.items) == 0
    def push(self, item):
        self.items.append(item)
    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        return None
    def peek(self):
        if not self.is_empty():
            return self.items[-1]
        return None
    def size(self):
        return len(self.items)
```

Test the Stack class

```
s = Stack()
print(s.is_empty()) # True
s.push(1)
s.push(2)
s.push(3)
print(s.peek())    # 3
print(s.size())    # 3
print(s.pop())     # 3
print(s.size())    # 2
print(s.is_empty()) # False
```

OutPut:

True

3

3

3

2

False

इनफिक्स ते पोस्टफिक्स रुपांतरण:

इनफिक्स एक्सप्रेसनला पोस्टफिक्स एक्सप्रेसनमध्ये रुपांतर करण्याचा कोड खाली दिला आहे:

```
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return len(self.items) == 0
    def push(self, item):
        self.items.append(item)
    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        return None
    def peek(self):
        if not self.is_empty():
            return self.items[-1]
        return None
def infix_to_postfix(expression):
    precedence = {'+': 1, '-': 1, '*': 2, '/': 2, '^': 3}
    stack = Stack()
    postfix = []
    for char in expression:
        if char.isalnum():
            postfix.append(char)
        elif char == '(':
            stack.push(char)
        elif char == ')':
            top = stack.pop()
            while top != '(':
                postfix.append(top)
            top = stack.pop()
        else:
            while (not stack.is_empty() and stack.peek() != '(' and
                   precedence[char] <= precedence[stack.peek()]):
                postfix.append(stack.pop())
            stack.push(char)
    while not stack.is_empty():
        postfix.append(stack.pop())
    return "".join(postfix)
# Example usage
expression = "A*(B+C)/D"
print("Infix:", expression)
```

```
print("Postfix:", infix_to_postfix(expression))
```

Output:

Infix: A*(B+C)/D

Postfix: ABC+*D/

इनफिक्स ते प्रिफिक्स रुपांतरण:

इनफिक्स एक्सप्रेसनला ला प्रिफिक्स एक्सप्रेसनलांमध्ये रुपांतर करण्याचा कोड खाली दिला आहे:

```
def infix_to_prefix(expression):
    precedence = {'+': 1, '-': 1, '*': 2, '/': 2, '^': 3}
    stack = Stack()
    prefix = []
    expression = expression[::-1]
    for char in expression:
        if char.isalnum():
            prefix.append(char)
        elif char == ')':
            stack.push(char)
        elif char == '(':
            top = stack.pop()
            while top != ')':
                prefix.append(top)
                top = stack.pop()
            else:
                while (not stack.is_empty() and stack.peek() != ')' and
                       precedence[char] < precedence[stack.peek()]):
                    prefix.append(stack.pop())
                stack.push(char)
            while not stack.is_empty():
                prefix.append(stack.pop())
            return ".join(prefix[::-1])

# Example usage
expression = "A*(B+C)/D"
print("Infix:", expression)
print("Prefix:", infix_to_prefix(expression))
```

Output :

Infix: A*(B+C)/D

Prefix: /*A+BCD

या कोडद्वारे तुम्ही इनफिक्स एक्सप्रेसनला पोस्टफिक्स आणि प्रिफिक्स एक्सप्रेसनमध्ये रुपांतर करू शकता. तुमच्याकडे कोणतेही प्रश्न असल्यास, कृपया सांगा!

5.5 क्यू: क्यूचा परिचय, क्यू अलिकेशन - ब्रेथ फर्स्ट सर्च, डेप्थ फर्स्ट सर्च (Queues: introduction to queues, queue applications - breadth first search, depth first search):

- **क्यू (Queue):** एक क्यू(Queue) हे एक डेटा संरचना आहे ज्यामध्ये प्रथम-दाखवा-प्रथम (FIFO) क्रमाने घटके जोडण्याची आणि काढण्याची परवानगी दिली जाते. अर्थात, queue मध्ये सर्वात पहिला घटक जोडला गेला तो सर्वात पहिल्यांचा काढला जाईल. या संरचनेचा वापर कार्य सारख्या आयामांच्या क्रमवारीत करण्यात येते, जसे की कार्य सूचीकरण, कार्य प्रक्रिया, आणि घटना व्यवस्थापन.

क्यू वापरण्याचे कारण (Reason of using Queue):

❖ क्रमवारीत ठेवणे:

- तो घटकांच्या क्रमाची ठेव ठेवते (ज्या घटके जोडायचे आहेत).
- घटके प्रक्रिया करण्याची क्रमवारीत फर्स्ट-इन, फर्स्ट-आऊट (FIFO) प्रक्रिया असते.

❖ टास्क शेड्यूलिंग आणि इवेन्ट हँडलिंग (Task Scheduling and event handling) :

- तो कार्य निर्धारित करण्यासाठी आणि प्राथमिकता देण्यासाठी मदत करते.
- उच्च प्राथमिकता असलेल्या कार्यांचे प्रक्रिया करण्यासाठी, कार्ये queue मध्ये जोडून आणि क्रमवारीत प्रक्रिया करून त्यांची खात्री करू शकता.

❖ मल्टी-थ्रेडिंग (Multithreading):

- मल्टी-थ्रेडिंग पर्यायांत्रित करण्यासाठी थ्रेड्सच्या दरम्यान डेटा प्रवाह समन्वय करण्यासाठी वापरले जाते.
- एक थ्रेड-सुरक्षित अंमलबजावणीच्या अंतर्गत, आपण अनेक थ्रेड्सच्या दरम्यान स्टॅकमध्ये घटके जोडून किंवा काढून सुरक्षितपणे करू शकता.

❖ असिन्क्रोनस प्रोग्रामिंग(Asynchronous Programming):

- विविध कार्यांच्या दरम्यान डेटा प्रवाह समन्वय करण्यासाठी मदत करते.
- डेटा स्टॅकमध्ये जोडत आणि असिन्क्रोनस प्रक्रिया करत असल्यामुळे आपले कार्य कार्यक्षम आणि प्रतिसादी ठेवते.

अशी मूलभूत वापरांमुळे पायथन मध्ये क्यू अंमलबजावणी अत्यंत उपयुक्त आणि अपरिहार्य आहे. पायथन मध्ये विविध प्रकारच्या क्यू आहेत, प्रत्येकाच्या वैशिष्ट्ये आणि वापरांसाठी वेगवेगळी प्रकारे आहेत. येथे काही उदाहरणे दिली गेली आहेत:

1 . सिम्पल क्यू (Simple Queue)

- ❖ या मूलभूत डेटा संरचनेमध्ये enqueue आणि dequeue कार्ये समर्थित केल्या जातात.
- ❖ घटके queue च्या शेवटी जोडली जातात आणि queue च्या प्रथमावरून काढले जातात, याचा FIFO (पहिलं जोडलं, पहिलं काढलं) क्रमानुसार वापर असतो.
- ❖ पायथन मध्ये हे lists, collections.deque, आणि queue.Queue वापरून अमलात येते.

2. प्रायोरिटी क्यू (Priority Queue):

- ❖ एक queue आहे ज्यामध्ये घटकांना त्यांच्या प्राथमिकता स्तरानुसार प्रक्रिया केली जाते.
- ❖ उच्च प्राथमिकता असलेला घटक खालील प्राथमिकता असलेल्या घटकापूर्वी प्रक्रिया करण्यात येतो.
- ❖ पायथन मध्ये हे heapq मॉड्यूल किंवा queue.PriorityQueue वापरून अमलात येते.

3. सर्क्युलर क्यू (Circular Queue):

- ❖ ही एक प्रकारची क्यू डेटा संरचना आहे, जी सामान्यतः रिअल-टाईम सिस्टममध्ये (जसे की बफरिंग, प्रिंट जॉब्स, शेड्युलिंग इत्यादी) वापरली जाते. साधारण क्यूमध्ये, जेव्हा क्यू भरले जाते, तेव्हा नवीन घटक जोडण्यासाठी क्यूमध्ये जागा नसते. परंतु सर्क्युलर क्यूमध्ये एक सर्क्युलर संरचना आहे, ज्यामध्ये क्यूच्या शेवटी नवीन घटक जोडल्यावर, जर क्यू पूर्णपणे भरले असेल, तर ते क्यूच्या सुरुवातीच्या जागी परत जातात..
- ❖ पायथन मध्ये हे एक निश्चित-आकारी सूची किंवा `collections.deque` वापरून अमलात येते.

4. डबल एंडेड क्यू (Deque Queue):

- ❖ Deque किंवा Double Ended Queue हे डेटा संरचना आहे ज्यामध्ये घटके विचारले आणि काढले जाऊ शकतात.
- ❖ पायथन मध्ये हे `collections.deque` वापरून अमलात येते.

5. लास्ट इन फर्स्ट ओउट क्यू (LIFO Queue):

- LIFO (किंवा Last-In, First-Out) queue हे डेटा संरचना आहे ज्यामध्ये अंतिम जोडलेला घटक प्रथम काढला जाईल.

पायथन मध्ये हे `queue.LifoQueue` वापरून अमलात येते.

यामध्ये नमुन्यात दिलेल्या प्रकारांपेक्षा अधिक प्रकारे असतात, आणि प्रत्येक प्रकारासाठी त्यांच्या विशेष वापरासाठी योग्यता आहे. पायथन वापरून ह्या विविध प्रकारच्या queues वापरून डेटा संरचना आणि अनुप्रयोगांमध्ये सुधारणा करण्यास मदत होते.

पायथनमध्ये क्यू कसा करावा (How to create Queue in Python)

1. लिस्ट वापरून (Using List)

पायथनमध्ये Queue अंमलबजावणी करण्याची एक सोपी पद्धत आहे ती असली तरी तुम्ही लिस्ट वापरू शकता. घटके लिस्टच्या शेवटीनं `append()` मेथडचा वापर करून जोडली जातात आणि प्रारंभीकडून `pop()` मेथडचा वापर करून index 0 वरून काढली जातात. उदाहरणासाठी:

```
# implement a queue using a list
```

```
queue = []
queue.append(1) # enqueue
queue.append(2)
queue.append(3)
print(queue)   # [1, 2, 3]
x = queue.pop(0) # dequeue
print(x)       # 1
print(queue)   # [2, 3]
```

2. `collections.deque` वापरून

पायथनच्या `collections` मॉड्यूलमध्ये `deque` (डबल-एंडेड क्यू) डेटा संरचना आहे, ज्याचा वापर करून तुम्ही डायरेक्शनली राइटवरून जोडण्यासाठी आणि लेफ्टवरून काढण्यासाठी क्यू म्हणून वापर करू शकता. उदाहरणस्वरूप:

```
# implement queue using collections.deque
from collections import deque
queue = deque()
queue.append(1) # enqueue
queue.append(2)
queue.append(3)
print(queue) # deque([1, 2, 3])
x = queue.popleft() # dequeue
print(x) # 1
print(queue) # deque([2, 3])
```

3.queue.Queue वापरून

पायथनच्या queue मॉड्यूलमध्ये 'Queue' वर्ग आहे, ज्याचा वापर मल्टीथ्रेडेड मध्ये वापरला जाऊ शकतो. घटके 'put()' मेथडचा वापर करून क्यूत घटक जोडल्या जाऊ शकतात आणि 'get()' मेथडचा वापर करून क्यूतून घटक काढल्या जाऊ शकतात. उदाहरणस्वरूप:

```
# implement queue using queue.Queue
import queue
queue = queue.Queue()
queue.put(1) # enqueue
queue.put(2)
queue.put(3)
print(queue.queue) # [1, 2, 3]
x = queue.get() # dequeue
print(x) # 1
print(queue.queue) # [2, 3]
```

❖ प्रगतीशील क्यू ऑपरेशन्स (Operation on Queue)

1. ब्लॉकिंग ऑपरेशन(Blocking Operation):

पायथनच्या queue मॉड्यूलमध्ये ब्लॉकिंग ऑपरेशन्स म्हणजेच get() आणि put() मेथड्स, जे मल्टीथ्रेडेड मध्ये एक थ्रेड क्यूमधून डेटा काढत असताना किंवा क्यूमध्ये डेटा जोडत असताना थ्रेडला ब्लॉक करतात, म्हणजेच थ्रेड थांबवला जातो जोपर्यंत अपेक्षित कार्य पूर्ण होत नाही.

2. क्यू साईज सीमा (Size of Queue):

maxsize पॅरामीटर (किव्हा maxsize() मेथड) वापरून कोणत्याही क्यू आकार सेट केले जाऊ शकते.क्यू आपल्या महत्त्वाच्या आकारात पोहोचल्यास, अधिक पुट() क्रियांना वारंवार वारंवार अवरोध करेल जेव्हा जागा उपलब्ध होते.याचा वापर ते क्यूने अधिक स्मृती वापरण्याची रोखणीसाठी उपयुक्त आहे.

2. क्यू जोइनिंग (Joining of Queue):

क्यूने queue.join() मेथड वापरून दोन किव्हा अधिक क्यू जोडता येतात, जेथे प्रत्येक घटक काढला जाण्यापर्यंत अवरोध करतो."

❖ आपण क्यूमध्ये खालील ऑपरेशन करू शकता.

इंक्यू Enqueue - Enqueue हा ऑपरेशन आहे जेथे आपण क्यूमध्ये घटके जोडता. जर क्यू पूर्ण असेल तर ते क्यूच्या अवस्था आहे. Enqueue कॉम्प्लेक्स सिटी $O(1)$ आहे.

डीक्यू Dequeue - Dequeue हा ऑपरेशन आहे जेथे आपण क्यूमध्ये एक घटक काढता. एक घटक त्याच्या समाविष्टीत त्याच्या व्यवस्थेत होतं. जर क्यू रिक्त असेल तर ते क्यूच्या अवस्था आहे. Dequeue चा कॉम्प्लेक्स सिटी $O(1)$ आहे.

फ्रॉन्ट Front - Front हा ऑपरेशन सुरवातीला एक घटक टाकतो. Front चा कॉम्प्लेक्स सिटी $O(1)$ आहे.

रेअर Rear - Rear हा ऑपरेशन पाठीवरून एक घटक काढतो. Rear कॉम्प्लेक्स सिटी $O(1)$ आहे.

1. ब्रेथ फर्स्ट सर्च (Breadth-First Search):

विशेषता: या एक ब्रेथ फर्स्ट शोध प्रकार आहे जो कोणत्याही संदेश किंवा संरचनेच्या सर्व संभाव्य स्थानांची शोधणी एक निश्चित नोडपासून सुरू करतो. त्यात, प्रत्येक स्तरातील सर्व नोडजला स्तरक्रमाने शोधले जाते जेणेकरून तो नोड जुळवला जातो आहे. त्यानंतर, त्या नोडजला त्याचे

उपयोग: याचा मुख्य वापर ग्राफ, नेटवर्क, डेटाबेस किंवा इतर डेटा संरचनांमध्ये संदेशांची शोधणी करण्यात येतो, ज्यात निश्चित संबंधीतता शोधणे महत्वाचे आहे.

ग्राफ शोध (Graph Search) उदाहरण:

```
from collections import deque
def bfs(graph, start):
    visited = set()
    queue = deque([start])
    while queue:
        node = queue.popleft()
        if node not in visited:
            visited.add(node)
            print(f'Visited Node: {node}')
            for neighbor in graph[node]:
                if neighbor not in visited:
                    queue.append(neighbor)
```

उदाहरण ग्राफ

```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
print("BFS search:")
bfs(graph, 'A')
```

OutPut:

BFS search:

Visited Node: A

Visited Node: B

Visited Node: C

Visited Node: D

Visited Node: E

Visited Node: F

2. डेप्थ फर्स्ट सर्च (Depth-First Search):

विशेषता: या एक डेप्थ फर्स्ट शोध प्रकार आहे जो कोणत्याही नोडपासून सुरू होतो आणि त्या नोडच्या शेवटच्या स्तर पर्यंत चिल्ड्रन नोड शोधतो, आणि नंतर त्यांच्या सर्व शोधणी चिल्ड्रन नोड ची करतो.

उपयोग: याचा मुख्य वापर माहितीच्या शोधामध्ये आहे जी हियरार्कल डेटा संरचनांमध्ये असते, जसे की , निश्चितीकरण, किंवा विशेष संदेशांची प्राप्ती.

ब्रेड्थ फर्स्ट सर्च (Breadth-First Search, BFS) आणि डेप्थ फर्स्ट सर्च (Depth-First Search, DFS) चा उपयोग करून खालील उदाहरण समजवले आहे.

बाइनरी ट्री (Binary Tree) उदाहरण:

```
class TreeNode:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key
# DFS शोध फंक्शन
def dfs(node):
    if node:
        print(f'Visited Node: {node.val}')
        dfs(node.left)
        dfs(node.right)
# उदाहरण बाइनरी ट्री
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
print("DFS search:")
dfs(root)
```

Output:

DFS search:

Visited Node: 1

Visited Node: 2

Visited Node: 4

Visited Node: 5

Visited Node: 3

❖ **सारांश:**

1. ब्रेथ फर्स्ट सर्च (BFS):

- ❖ लेव्हल बाय लेव्हल ट्रॅव्हर्सल
- ❖ क्यू (queue) डेटा स्ट्रक्चर वापरतो

2. डेप्थ फर्स्ट सर्च (DFS):

- ❖ ब्रांच बाय ब्रांच ट्रॅव्हर्सल
- ❖ स्टॅक (stack) डेटा स्ट्रक्चर वापरतो
- ❖ दोन प्रकार: Recursive आणि Iterative

वरील उदाहरणांमधून तुम्ही BFS आणि DFS अल्गोरिदम कसे इम्प्लीमेंट करायचे ते शिकू शकता. तुम्ही आपल्या आवश्यकतेनुसार हे अल्गोरिदम बेस्ट काम करू शकतो .

संदर्भ सूची (References) :

1. <https://www.geeksforgeeks.org/how-to-use-lists-as-stacks-and-queues-in-python/>
2. <https://www.geeksforgeeks.org/array-data-structure-guide/>
3. <https://www.datacamp.com/tutorial/data-structures-guide-python>
4. <https://www.youtube.com/watch?v=pkYVOMU3MgA>
5. <https://loungecoder.com/dsa-in-python-linear-lists-and-arrays/>
6. <https://www.pythonforbeginners.com/data-types/stack-in-python>

युनिट -6

नॉन-लिनियर डेटा स्ट्रक्चर

(Non Linear Data Structure)

विषय निष्पत्ती (Course Outcome): ट्री डेटा स्ट्रक्चर लागू करण्यासाठी पायथन प्रोग्राम विकसित करा.

घटक निष्पत्ती (Theory Learning Outcome):

1. पायथन कोड वापरून विविध प्रकारच्या ट्री डेटा स्ट्रक्चरची अंमलबजावणी करणे.
2. पायथन कोड वापरून विविध प्रकारच्या ट्री डेटा स्ट्रक्चरवर विविध ऑपरेशन्स करा.

6.1 ट्री डेटा स्ट्रक्चर शब्दावली (Tree Terminology)

ट्री डेटा स्ट्रक्चर एक महत्वाची संकल्पना आहे जी संगणक विज्ञानात डेटा व्यवस्थापनासाठी वापरली जाते. हे एक हायरेरिकल स्वरूप आहे जिथे डेटा नोड्सच्या रूपात साठवला जातो आणि त्यांच्यात संबंध असतात. यात मध्यवर्ती नोड, स्ट्रक्चरल नोड्स आणि सब-नोड्स असतात, जे एजद्वारे जोडलेले असतात. आम्ही असेही म्हणू शकतो की ट्री च्या डेटा स्ट्रक्चरमध्ये, रूट एजेस आणि लीफ नोडजोडलेली आहेत.

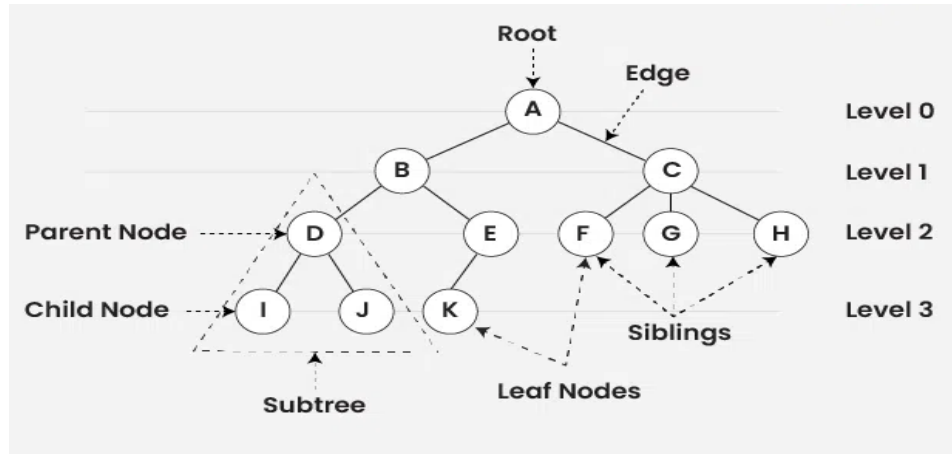


Figure 6.1: Tree data structure

6.1.1 ट्री डेटा स्ट्रक्चरचे घटक (Element of Tree Data Structure):

1. **रूट (Root):** ट्रीचा पहिला नोड जो सर्वात वर असतो.
2. **नोड (Node):** प्रत्येक पॉइंट जो डेटा साठवतो.
3. **एज (Edge):** दोन नोड्समध्ये संबंध असतो.
4. **लीफ (Leaf):** जो नोड कोणतेही चाइल्ड नसतो.
5. **चाइल्ड (Child):** नोडचा अनुक्रमीत नोड.
6. **पेरेंट (Parent):** एका किंवा अधिक नोडचा मूळ नोड.
7. **पेरेंट नोड (Parent Node):** नोड जो नोडचा पूर्ववर्ती आहे त्याला त्या नोडचा पेरेंट नोड म्हणतात. {B} हा {D, E} चा मूळ नोड आहे.
8. **चाइल्ड नोड (Child Node):** नोडचा तात्काळ उत्तराधिकारी असलेल्या नोडला त्या नोडचा चाइल्ड नोड म्हणतात. उदाहरणे: {D, E} हे {B} चे चाइल्ड नोड आहेत.

9. **रूट नोड (Root Node):** झाडाचा सर्वात वरचा नोड किंवा ज्या नोडमध्ये कोणतेही पॅरेंट नोड नाही त्याला रूट नोड म्हणतात. {A} झाडाचा मूळ नोड आहे. रिकामे नसलेल्या झाडामध्ये एक रूट नोड आणि मुळापासून झाडाच्या इतर सर्व नोड्सपर्यंत एक मार्ग असणे आवश्यक आहे.
10. **लीफ नोड किंवा एक्सटर्नल नोड (Leaf or external Node):** ज्या नोड्समध्ये चाइल्ड नोड्स नसतात त्यांना लीफ नोड्स म्हणतात. {K, L, M, N, O, P, G} झाडाच्या पानांच्या गाठी आहेत.
11. **नोडचा पूर्वज (Ancestor of a Node):** त्या नोडच्या रूटच्या मार्गावरील कोणत्याही पूर्ववर्ती नोड्सला त्या नोडचे पूर्वज म्हणतात. {A,B} हे नोड {E} चे पूर्वज नोड आहेत
12. **सिबलिंग (Sibling):** समान पालक नोडच्या मुलांना सिबलिंग म्हणतात. {D,E} यांना सिबलिंग म्हणतात.
13. **नोडची पातळी (Level of Node):** रूट नोडपासून त्या नोडपर्यंतच्या मार्गावरील कडांची गणना. रूट नोडमध्ये पातळी 0 आहे.
14. **अंतर्गत नोड (Internal Node):** कमीतकमी एक मूल असलेल्या नोडला अंतर्गत नोड म्हणतात.
15. **नोडचा शेजारी (Descendant):** त्या नोडच्या पालक किंवा मुलाच्या नोड्सना त्या नोडचे शेजारी म्हणतात.
16. **सबनोड (Sub Node):** झाडाचा कोणताही नोड त्याच्या वंशजांसह.

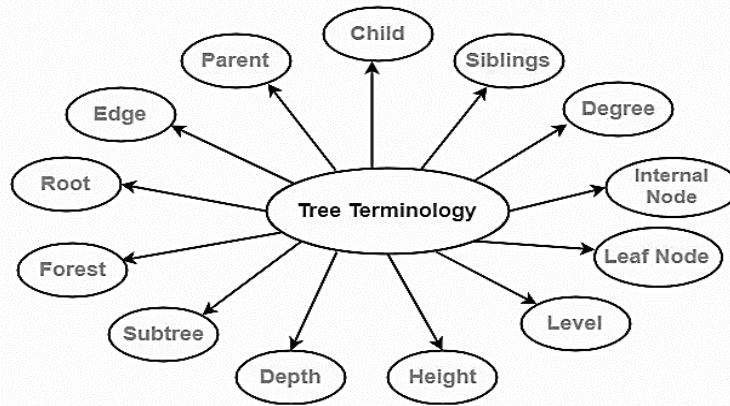


Figure 6.2: Tree Terminology

• ट्रीला नॉन-लिनियर डेटा स्ट्रक्चर का मानले जाते?

ट्री मध्ये डेटा अनुक्रमिक रीतीने संग्रहित केला जात नाही. त्याऐवजी, ते अनेक स्तरांवर व्यवस्थित संग्रहित केले जातात किंवा आपण म्हणू शकतो की ही एक श्रेणीबद्ध रचना आहे. या कारणास्तव, ट्रीला नॉन-लाइनर डेटा स्ट्रक्चर मानले जाते.

• ट्री डेटा स्ट्रक्चरचे Representative:

ट्रीमध्ये रूट नोड, आणि शून्य किंवा अधिक सबट्री $T_1, T_2, \dots, T_k$ असतात की ट्रीच्या रूट नोडपासून प्रत्येक सबनोडच्या रूट नोडपर्यंत एक एज असते. नोड X च्या सबट्रीमध्ये सर्व नोड्स असतात ज्यात रूट नोड म्हणून नोड X असतो.

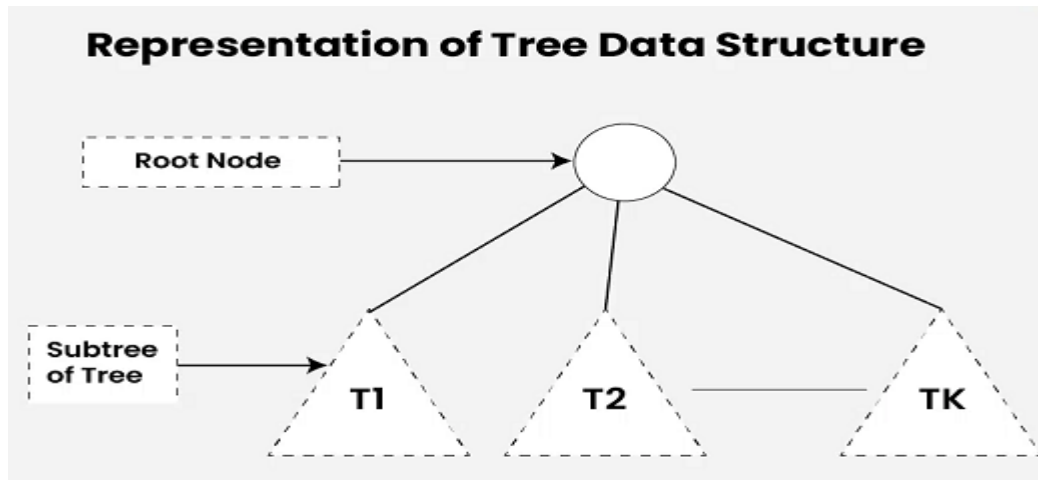


Figure 6.3: Tree data structure Representation

Python function to represent tree data structure

```

class TreeNode:
    def __init__(self, value):
        self.value = value
        self.children = []
    def add_child(self, child_node):
        self.children.append(child_node)
    def __repr__(self):
        return f"TreeNode({self.value})"

def tree_to_english(node, depth=0):
    indent = ' ' * depth
    representation = f"{indent}- {node.value}\n"
    for child in node.children:
        representation += tree_to_english(child, depth + 1)
    return representation

# Example usage
if __name__ == "__main__":
    root = TreeNode("Root")
    child1 = TreeNode("Child 1")
    child2 = TreeNode("Child 2")
    child1_1 = TreeNode("Child 1.1")
    child1_2 = TreeNode("Child 1.2")
    child2_1 = TreeNode("Child 2.1")
    root.add_child(child1)
    root.add_child(child2)
    child1.add_child(child1_1)
    child1.add_child(child1_2)
    child2.add_child(child2_1)
  
```

```
print(tree_to_english(root))
```

Output

When you run this code, it will produce a structured representation of the tree:

- Root
 - Child 1
 - Child 1.1
 - Child 1.2
 - Child 2
 - Child 2.1

6.1.2 ट्रीचे प्रकार (Type of Tree)

1. बायनरी ट्री (Binary Tree): प्रत्येक नोडला जास्तीत जास्त दोन चाइल्ड्स असतात. बायनरी ट्रीमध्ये, प्रत्येक नोडला जास्तीत जास्त दोन नोड जोडलेली असू शकतात. बायनरी ट्रीच्या काही सामान्य प्रकारांमध्ये पूर्ण बायनरी ट्री, पूर्ण बायनरी ट्री, संतुलित बायनरी ट्री आणि डिजनरेट किंवा पॅथॉलॉजिकल बायनरी ट्री यांचा समावेश होतो.

2. बायनरी सर्च ट्री (Binary search Tree): बायनरी ट्रीचा एक विशेष प्रकार जिथे प्रत्येक नोडचा डावीकडचा सबनोड छोटा आणि उजवीकडचा सबनोड मोठा असतो.

3. एविएल ट्री (AVL Tree): बायनरी सर्च ट्रीचा एक प्रकार जो संतुलित असतो.

4. बी-ट्री (B Tree): हे स्वतःला संतुलित करणारे ट्री आहे जे विशेषतः डेटाबेस आणि फाइल सिस्टममध्ये वापरले जाते.

5. टर्नरी ट्री (Turnary Tree): टर्नरी ट्री ही ट्री डेटा स्ट्रक्चर आहे ज्यामध्ये प्रत्येक नोडमध्ये जास्तीत जास्त तीन चाइल्ड नोड्स असतात, सामान्यतः "डावीकडे", "मध्य" आणि "उजवीकडे" असे तीन नोड ओळखले जातात.

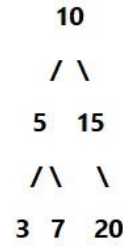
6. जेनेरिक ट्री (Generic Tree): जेनेरिक ट्री हे नोड्सचे संग्रह आहेत जिथे प्रत्येक नोड ही डेटा स्ट्रक्चर असते ज्यामध्ये रेकॉर्ड आणि त्याच्या चाइल्ड च्या संदर्भाची सूची. लिंक केलेल्या सूचीच्या विपरीत, प्रत्येक नोड एकाधिक नोड्सचा अड्रेस संग्रहित करतो.

• ट्रीचे उपयोग (Use of Tree):

1. डेटा सर्चिंग (Data Searching): बायनरी सर्च ट्रीमध्ये डेटा जलद शोधता येतो.
2. डेटा स्टोरेज (Data Storage): हायरेरिकल डेटा व्यवस्थित ठेवण्यासाठी.
3. डायनामिक सेट्स (Dynamic Sets): इन्सर्शन, डिलीशन आणि सर्चिंग सहजतेने करण्यासाठी.
4. डेटाबेस इंडेक्सिंग (Database Indexing): बी-ट्रीज डेटाबेसमध्ये जलद अॅक्सेससाठी वापरतात.

• ट्रीची बेसिक ऑपरेशन्स (Basic Operations on Tree):

1. इन्सर्ट (Insert): ट्रीमध्ये नवीन नोड इन्सर्ट करणे.
2. डिलीट (Delete): ट्रीमधून नोड काढणे.
3. सर्च (Search): ट्रीमध्ये एखाद्या नोडचा शोध घेणे.
4. ट्रव्हर्सल : सर्व नोड्सना एका विशिष्ट क्रमाने ट्रव्हर्सणे. (जसे इनऑर्डर, प्रीऑर्डर, पोस्टऑर्डर)

उदाहरण:

वरील उदाहरणात, 10 हे रूट नोड आहे. 5 आणि 15 हे 10 चे चाइल्ड्स आहेत. 5 चे चाइल्ड्स 3 आणि 7 आहेत, आणि 15 चा चाइल्ड 20 आहे.

ट्री डेटा संरचना डेटाच्या कार्यक्षम व्यवस्थापनासाठी अत्यंत प्रभावी आहे आणि संगणक विज्ञानातील विविध अनुप्रयोगांमध्ये वापरली जाते.

6.2 बायनरी ट्री

बायनरी ट्री एक प्रकारचा डेटा संरचना आहे ज्यामध्ये प्रत्येक नोडला जास्तीत जास्त दोन चाइल्ड नोड्स असतात, जे "डावा चाइल्ड" आणि "उजवा चाइल्ड" म्हणून ओळखले जातात. बायनरी ट्रीमध्ये डेटा हायरेरिकल पद्धतीने साठवला जातो आणि त्यात शोध, इन्सर्शन, डिलीशन, इत्यादी ऑपरेशन्स सुलभ असतात.

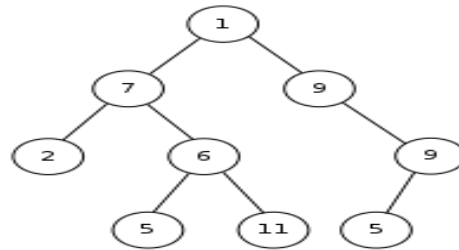


Figure 6.4: Binary tree

simple binary tree structure:

```
class TreeNode:
```

```
    def __init__(self, value):
```

```
        self.value = value
```

```
        self.left = None
```

```
        self.right = None
```

बायनरी ट्रीचे घटक (Element of Binary Tree):

1. **रूट (Root)** : ट्रीचा पहिला नोड जो सर्वात वर असतो.
2. **नोड (Node)**: प्रत्येक पॉइंट जो डेटा साठवतो.
3. **एज (Edge)**: दोन नोड्समध्ये संबंध असतो.
4. **लीफ (Leaf)**: जो नोड कोणतेही चाइल्ड नसतो.
5. **चाइल्ड (Child)**: नोडचा अनुक्रमीत नोड.
6. **पेरेंट (Parent)**: एका किंवा अधिक सबनोड चा रूट नोड.

7. **सिबलिंग (Sibling) :** समान पेरेंट असलेल्या नोड्स.

बायनरी ट्री ऑपरेशन्स (Operations on Binary Tree):

1. **इन्सर्शन (Insertion):** ट्रीमध्ये नवीन नोड घालणे.
2. **डिलीट (Deletion):** ट्रीमधून नोड काढणे.
3. **सर्च (Search):** ट्रीमध्ये एखाद्या नोडचा शोध घेणे.
4. **ट्राव्हर्सल (Traversal):** सर्व नोड्सना एका विशिष्ट क्रमाने ट्रव्हर्सणे.
 - a. **इनऑर्डर ट्राव्हर्सल (Inorder Traversal):** लेफ्ट - रूट - राईट
 - **प्रीऑर्डर ट्राव्हर्सल (Preorder Traversal):** रूट - लेफ्ट - राईट
 - **पोस्टऑर्डर ट्राव्हर्सल (Postorder Traversal):** लेफ्ट - राईट - रूट

बायनरी ट्रीचे उदाहरण:

```

      8
     /\
    3 10
   /\  \
  1 6  14
   /\ /
  4 7 13

```

वरील उदाहरणात:

- 8 हे रूट नोड आहे.
- 8 चे चाइल्ड्स 3 आणि 10 आहेत.
- 3 चे चाइल्ड्स 1 आणि 6 आहेत.
- 6 चे चाइल्ड्स 4 आणि 7 आहेत.
- 10 चा चाइल्ड 14 आहे.
- 14 चा चाइल्ड 13 आहे.

6.2.1 अरे वापरून बायनरी ट्रीचे प्रतिनिधित्व (Representation of Binary Tree using array)

बायनरी ट्रीची अरे मध्ये प्रतिनिधित्व करण्याची पद्धत सोपी आणि प्रभावी आहे, विशेषतः पूर्णांकित (Complete) बायनरी ट्रीसाठी. अरे आधारित प्रतिनिधित्वात, बायनरी ट्रीमधील नोड्स एका अरेमध्ये सलगपणे ठेवले जातात. अरेमध्ये नोड्सची अनुक्रमणिका विशिष्ट पद्धतीने ठेवली जाते ज्यामुळे नोड्सचे पेरेंट्स आणि चाइल्ड्स सहजपणे शोधता येतात.

अरेमध्ये बायनरी ट्रीचे प्रतिनिधित्व:

उदाहरण:

एका साध्या बायनरी ट्रीचे उदाहरण घेऊ:

```

      1
     /\
    2 3
   /\ /\
  4 5 6 7

```

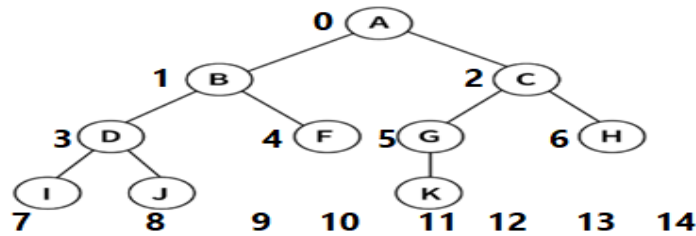
वरील बायनरी ट्रीचे अरे प्रतिनिधित्व असे होईल:

अनुक्रमणिका	मूल्य
0	1
1	2
2	3
3	4
4	5
5	6
6	7

अरेची संरचना (Structure of Array):

A = [1, 2, 3, 4, 5, 6, 7]

- **रूट नोड (1)** अरेच्या 0व्या स्थानावर आहे.
- **नोड 2** चे लेफ्ट चाइल्ड (4) अरेच्या 3व्या स्थानावर आहे आणि राइट चाइल्ड (5) अरेच्या 4व्या स्थानावर आहे.
- **नोड 3** चे लेफ्ट चाइल्ड (6) अरेच्या 5व्या स्थानावर आहे आणि राइट चाइल्ड (7) अरेच्या 6व्या स्थानावर आहे.
- **उदाहरण**



वरील ट्री अरे प्रतिनिधित्व(T):

'A'	'B'	'C'	'D'	'F'	'G'	'H'	'I'	'J'			'K'			
T[0]	T[1]	T[2]	T[3]	T[4]	T[5]	T[6]	T[7]	T[8]	T[9]	T[10]	T[11]	T[12]	T[13]	T[14]

Figure 6.5: Binary tree representation using array

6.2.2 बायनरी ट्रीचे लिंक केलेले प्रतिनिधित्व (Linked Representation of Binary Tree)

लिंकड प्रतिनिधित्व (Linked Representation) मध्ये बायनरी ट्री प्रत्येक नोडला एक स्वतंत्र डेटा स्ट्रक्चर म्हणून साठवले जाते, ज्यामध्ये प्रत्येक नोडमध्ये डेटा आणि दोन पॉइंटर्स (डावा आणि उजवा चाइल्ड कडे निर्देशित करणारे) असतात. ही पद्धत विशेषतः बायनरी ट्रीची ट्रेवर्सल आणि मॅनिपुलेशन सुलभ करते.

बायनरी ट्रीचे लिंकड प्रतिनिधित्व- Linked Representation

प्रत्येक नोडचे एक साधे साधे (Structure) असेल ज्यामध्ये तीन घटक असतील:

1. डेटा (Data): नोडचे मूल्य.
2. डावा चाइल्ड (Left Child): डावा सबनोड निर्देशित करणारा पॉइंटर.
3. उजवा चाइल्ड (Right Child): उजवा सबनोड निर्देशित करणारा पॉइंटर.

बायनरी ट्रीचे उदाहरण (Example of Binary Tree):

लिंकड प्रतिनिधित्व (Linked Representation):

प्रत्येक नोडचे लिंकड प्रतिनिधित्व खालीलप्रमाणे असेल: Python मध्ये लिंकड लिस्ट वापरून ट्री डेटा स्ट्रक्चरचे प्रतिनिधित्व केले जाते. `TreeNode` वर्ग जेथे प्रत्येक नोडचे मूल्य आणि चाइल्ड ची लिंक केलेली सूची असते.

खालीलप्रमाणे केले जाते:

1. **Define the ListNode class** for linked list nodes.
2. **Define the TreeNode class** that uses ListNode for its children.
3. **Create a function** to represent the tree

उदाहरण : ट्री खालीलप्रमाणे लिंक केलेल्या सूचीचा वापर करून प्रस्तुत केले जाऊ शकते.

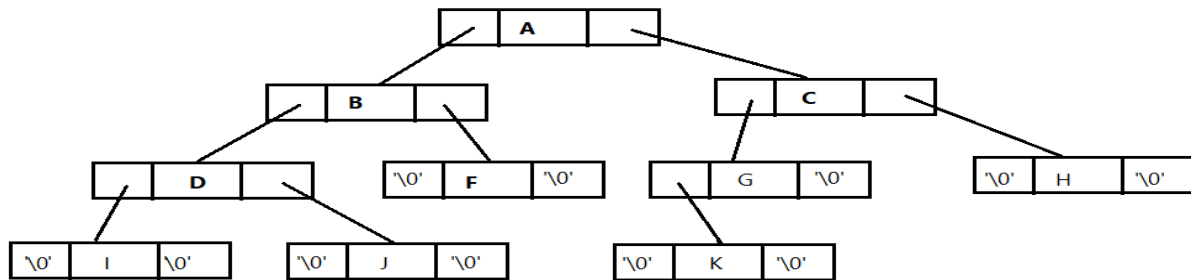


Figure 5.6: Linked binary tree Representation

Code

class ListNode:

```

def __init__(self, tree_node=None, next_node=None):
    self.tree_node = tree_node
    self.next_node = next_node

```

class TreeNode:

```

def __init__(self, value):
    self.value = value
    self.children = None # Head of the linked list of children

def add_child(self, child_node):
    if not self.children:

```

```

        self.children = ListNode(child_node)
    else:
        current = self.children
        while current.next_node:
            current = current.next_node
        current.next_node = ListNode(child_node)
def tree_to_english(node, depth=0):
    indent = ' ' * depth
    representation = f"{indent}- {node.value}\n"
    current = node.children
    while current:
        representation += tree_to_english(current.tree_node, depth + 1)
        current = current.next_node
    return representation
# Example usage
if __name__ == "__main__":
    root = TreeNode("Root")
    child1 = TreeNode("Child 1")
    child2 = TreeNode("Child 2")
    child1_1 = TreeNode("Child 1.1")
    child1_2 = TreeNode("Child 1.2")
    child2_1 = TreeNode("Child 2.1")
    root.add_child(child1)
    root.add_child(child2)
    child1.add_child(child1_1)
    child1.add_child(child1_2)
    child2.add_child(child2_1)
    print(tree_to_english(root))

```

नोड ऑपरेशन्स (Node Operations):

1. इन्सर्शन (Insertion):

- लिंकड प्रतिनिधित्वात, नवीन नोड इन्सर्शन म्हणजे नवीन नोड तयार करणे आणि योग्य ठिकाणी पॉइंटर्स समायोजित करणे.

2. डिलीट (Deletion):

- डिलीट करताना, त्या नोडला पॉइंटर्स बदलून काढणे आवश्यक असते आणि उर्वरित नोड्स पुनर्संयोजन करणे आवश्यक असते.

3. सर्च (Search):

- ट्री ट्रेव्हर्सलच्या विविध पद्धतींनी (जसे इनऑर्डर, प्रीऑर्डर, पोस्टऑर्डर) आवश्यक नोडचा शोध घेता येतो.

4. ट्रेव्हर्सल (Traversal):

- बायनरी ट्री ट्रेव्हर्सल करताना तीन मुख्य प्रकार वापरले जातात:
 - इनऑर्डर ट्राव्हर्सल (Inorder Traversal): लेफ्ट - रूट - राईट
 - प्रीऑर्डर ट्राव्हर्सल (Preorder Traversal): रूट - लेफ्ट - राईट
 - पोस्टऑर्डर ट्राव्हर्सल (Postorder Traversal): लेफ्ट - राईट - रूट

1. इनऑर्डर ट्रेव्हर्सलचे function (Python मध्ये):

```
def inorder_traversal(node, result=None):
    if result is None:
        result = []
    if node:
        inorder_traversal(node.left, result) # Visit left subtree
        result.append(node.value)           # Visit node itself
        inorder_traversal(node.right, result) # Visit right subtree
    return result
```

वरील उदाहरणामध्ये, ट्रीचे इनऑर्डर ट्रेव्हर्सल हे सर्व नोड्सला डावीकडून उजवीकडे ट्रेव्हर्स आणि परिणाम स्वरूप प्रिंट करते. लिंकड प्रतिनिधित्व बायनरी ट्रीच्या विविध ऑपरेशन्स सुलभ करते आणि हे संरचना विशेषत: जटिल डेटा संरचना आणि अल्गोरिदमसाठी सबयोगी आहे.

2. प्रीऑर्डर ट्रेव्हर्सल function (Python मध्ये):

```
def preorder_traversal(node, result=None):
    if result is None:
        result = []
    if node:
        result.append(node.value) # Visit node itself
        preorder_traversal(node.left, result) # Visit left subtree
        preorder_traversal(node.right, result) # Visit right subtree
    return result
```

3. पोस्टऑर्डर ट्रेव्हर्सल function (Python मध्ये):

```
def postorder_traversal(node, result=None):
    if result is None:
        result = []
    if node:
        postorder_traversal(node.left, result) # Visit left subtree
        postorder_traversal(node.right, result) # Visit right subtree
        result.append(node.value) # Visit node itself
    return result
```

6.2.3 बायनरी ट्रीचे प्रकार (Types of Binary Tree)

बायनरी ट्रीचे विविध प्रकार आहेत, प्रत्येकाचे विशिष्ट गुणधर्म आणि सबयोग आहेत. खालीलप्रमाणे मुख्य बायनरी ट्री प्रकारांबद्दल माहिती दिली आहे:

1. पूर्ण बायनरी ट्री (Full Binary Tree)

- पूर्ण बायनरी ट्रीमध्ये प्रत्येक नोडला दोन किंवा शून्य चाइल्ड असतात. म्हणजेच, प्रत्येक नोड किंवा तर दोन चाइल्ड्स असतात किंवा ते एक लीफ नोड असतात.

उदाहरण:

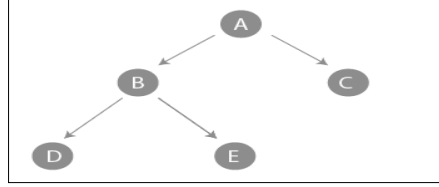


Figure 6.7: Full Binary Tree

2. परिपूर्ण बायनरी ट्री (Perfect Binary Tree)

- परिपूर्ण बायनरी ट्रीमध्ये सर्व इंटरनल नोड्सना दोन चाइल्ड्स असतात आणि सर्व लीफ नोड्स समान लेवलवर असतात.

उदाहरण:

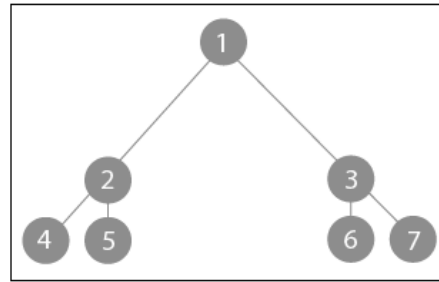


Figure 6.8: Perfect Binary Tree

3. पूर्णांकित बायनरी ट्री (Complete Binary Tree)

- पूर्णांकित बायनरी ट्रीमध्ये, शेवटच्या लेवल वगळता सर्व लेवल पूर्णपणे भरलेल्या असतात, आणि शेवटची लेवल डावीकडून उजवीकडे भरलेली असते.

उदाहरण:

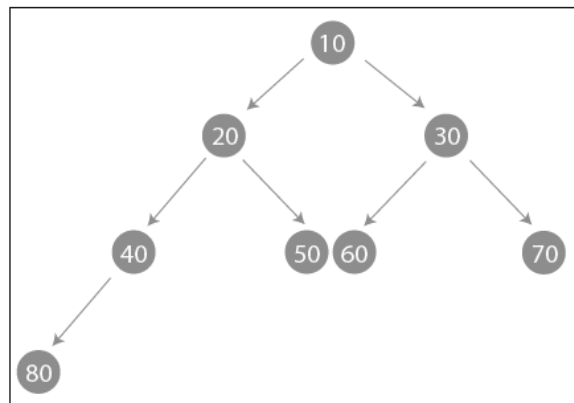


Figure 6.9: Complete Binary Tree

4. बायनरी सर्च ट्री (Binary Search Tree)

- बायनरी सर्च ट्री (BST) एक विशेष प्रकारचा बायनरी ट्री आहे ज्यामध्ये प्रत्येक नोडची अशी रचना असते की डावीकडचा सबनोड (subtree) त्या नोडपेक्षा कमी मूल्याचे असतो आणि उजवीकडचा सबनोड त्या नोडपेक्षा जास्त मूल्याचे असतो.

उदाहरण:

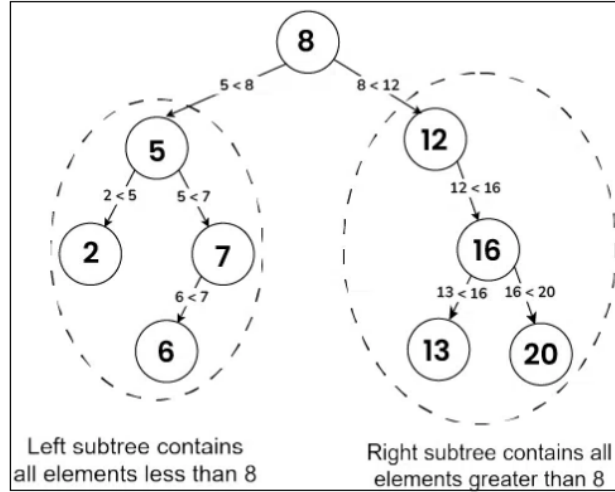


Figure 6.10: Binary Search Tree

5. स्क्युड बायनरी ट्री (Skewed Binary Tree):

- स्क्युड बायनरी ट्री हा बायनरी ट्रीचा एक प्रकार आहे.
- स्क्युड ट्री दोन विशेष प्रकार आहेत
 - लेफ्ट स्क्युड बायनरी ट्री (Left Skewed Binary Tree):** ही अशी स्क्युड बायनरी ट्री आहेत ज्यात सर्व नोड्समध्ये डावे चाइल्ड असते. ही डाव्या बाजूचे वर्चस्व असलेली ट्री आहे.
 - उजवे स्क्युड बायनरी ट्री (Right Skewed Binary Tree):** ही अशी स्क्युड बायनरी ट्री आहेत ज्यात सर्व नोड्सना योग्य चाइल्ड आहे किंवा चाइल्डच नाही. ही उजव्या बाजूचे वर्चस्व असलेली ट्री आहे.

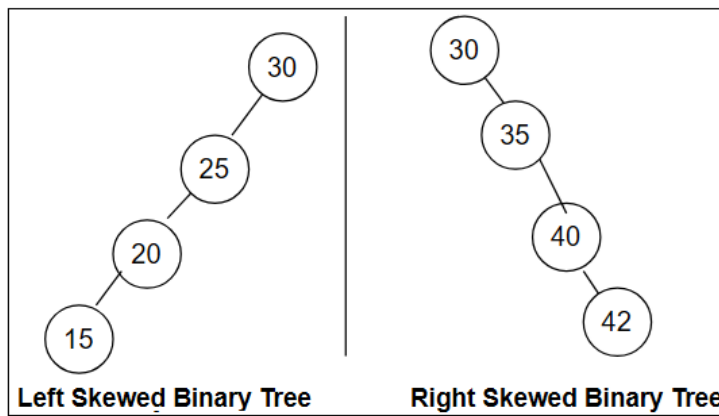


Figure 6.11: Skewed Binary Tree

या विविध प्रकारच्या बायनरी ट्रीजमुळे विविध संगणकीय समस्यांवर प्रभावी आणि कार्यक्षम उपाय शोधणे सोपे होते. प्रत्येक प्रकारचा बायनरी ट्री विशिष्ट अनुप्रयोगांमध्ये वापरला जातो आणि त्यांच्या विशेष गुणधर्मांमुळे डेटा साठवणे, शोधणे, आणि व्यवस्थापित करणे सुलभ होते.

6.3 बायनरी ट्री ट्रॅव्हर्सल (Binary Tree Traversal)

बायनरी ट्री ट्रॅव्हर्सल म्हणजे एका विशिष्ट क्रमाने बायनरी ट्रीमधील सर्व नोड्सना ट्रॅव्हर्स देण्याची प्रक्रिया. बायनरी ट्री ट्रॅव्हर्सलच्या काही सामान्य पद्धती आहेत:

- **इनऑर्डर ट्रॅव्हर्सल (Inorder Traversal):** लेफ्ट - रूट - राईट
- **प्रीऑर्डर ट्रॅव्हर्सल (Preorder Traversal):** रूट - लेफ्ट - राईट
- **पोस्टऑर्डर ट्रॅव्हर्सल (Postorder Traversal):** लेफ्ट - राईट - रूट
- **इन-ऑर्डर ट्रॅव्हर्सल**

1. इनऑर्डर ट्रॅव्हर्सल (Inorder Traversal): इन-ऑर्डर ट्रॅव्हर्सलमध्ये नोड्सला लेफ्ट - रूट - राईट या क्रमाने ट्रॅव्हर्स दिली जाते. याचा अर्थ, सर्वप्रथम डाव्या सबट्रीला ट्रॅव्हर्स दिली जाते, नंतर रूट नोडला आणि शेवटी उजव्या सबट्रीला.

अल्गोरिदम:

1. डाव्या सबट्रीला ट्रॅव्हर्स करा.
2. रूट नोडला ट्रॅव्हर्स करा.
3. उजव्या सबट्रीला ट्रॅव्हर्स करा.

Python Implementation:

```
class Node:
```

```
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key
```

```
def in_order_traversal(root):
```

```
    if root:
        in_order_traversal(root.left)
        print(root.val, end=' ')
        in_order_traversal(root.right)
```

Example :

```
# root = Node(1)
# root.left = Node(2)
# root.right = Node(3)
# root.left.left = Node(4)
# root.left.right = Node(5)
# in_order_traversal(root)
```

Output: 4 2 5 1 3

2. प्री-ऑर्डर ट्रव्हर्सल

प्री-ऑर्डर ट्रव्हर्सलमध्ये नोड्सला रूट - लेफ्ट - राईट या क्रमाने ट्रव्हर्स दिली जाते. सर्वप्रथम रूट नोडला ट्रव्हर्स दिली जाते, नंतर डाव्या सबट्रीला आणि शेवटी उजव्या सबट्रीला.

अल्गोरिदम:

1. रूट नोडला ट्रव्हर्स करा.
2. डाव्या सबट्रीला ट्रव्हर्स करा.
3. उजव्या सबट्रीला ट्रव्हर्स करा.

Python Implementation:

```
def pre_order_traversal(root):
```

```
    if root:
```

```
        print(root.val, end=' ')
```

```
        pre_order_traversal(root.left)
```

```
        pre_order_traversal(root.right)
```

```
# Example:
```

```
# pre_order_traversal(root) # Output: 1 2 4 5 3
```

3. पोस्ट-ऑर्डर ट्रव्हर्सल

पोस्ट-ऑर्डर ट्रव्हर्सलमध्ये नोड्सला लेफ्ट - राईट - रूट या क्रमाने ट्रव्हर्स दिली जाते. प्रथम डाव्या सबट्रीला, नंतर उजव्या सबट्रीला आणि शेवटी रूट नोडला ट्रव्हर्स दिली जाते.

अल्गोरिदम:

1. डाव्या सबट्रीला ट्रव्हर्स करा.
2. उजव्या सबट्रीला ट्रव्हर्स करा.
3. रूट नोडला ट्रव्हर्स करा.

Python Implementation:

```
def post_order_traversal(root):
```

```
    if root:
```

```
        post_order_traversal(root.left)
```

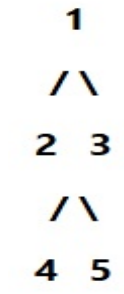
```
        post_order_traversal(root.right)
```

```
        print(root.val, end=' ')
```

```
# Example:
```

```
# post_order_traversal(root) # Output: 4 5 2 3 1
```

उदाहरणे:



या ट्रीमध्ये:

- रूट नोड 1 आहे.
- 1 च्या डाव्या बाजूला 2 आणि उजव्या बाजूला 3 आहे.
- 2 च्या डाव्या बाजूला 4 आणि उजव्या बाजूला 5 आहे.

1. इन-ऑर्डर ट्राव्हर्सल (लेफ्ट - रूट - राईट)

Traversal Order: 4 2 5 1 3

2. प्री-ऑर्डर ट्राव्हर्सल (रूट - लेफ्ट - राईट)

Traversal Order: 1 2 4 5 3

3. पोस्ट-ऑर्डर ट्राव्हर्सल (लेफ्ट - राईट - रूट)

Traversal Order: 4 5 2 3 1

Python कोड उदाहरण

ट्री तयार करण्यासाठी आणि विविध ट्राव्हर्सल्स करण्यासाठी पायथन कोड:

class Node:

```
def __init__(self, key):
    self.left = None
    self.right = None
    self.val = key
```

def in_order_traversal(root):

```
    if root:
        in_order_traversal(root.left)
        print(root.val, end=' ')
        in_order_traversal(root.right)
```

def pre_order_traversal(root):

```
    if root:
        print(root.val, end=' ')
        pre_order_traversal(root.left)
        pre_order_traversal(root.right)
```

def post_order_traversal(root):

```
    if root:
        post_order_traversal(root.left)
        post_order_traversal(root.right)
        print(root.val, end=' ')
```

from collections import deque

def level_order_traversal(root):

```
    if not root:
        return
    queue = deque([root])
    while queue:
        node = queue.popleft()
```



```

print(node.val, end=' ')
    if node.left:
        queue.append(node.left)
    if node.right:
        queue.append(node.right)
# Example usage:
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
print("In-order Traversal: ")
in_order_traversal(root) # Output: 4 2 5 1 3
print("\nPre-order Traversal: ")
pre_order_traversal(root) # Output: 1 2 4 5 3
print("\nPost-order Traversal: ")
post_order_traversal(root) # Output: 4 5 2 3 1
print("\nLevel-order Traversal: ")
level_order_traversal(root) # Output: 1 2 3 4 5

```

1. Traverse the following binary tree into preorder, postorder and inorder.

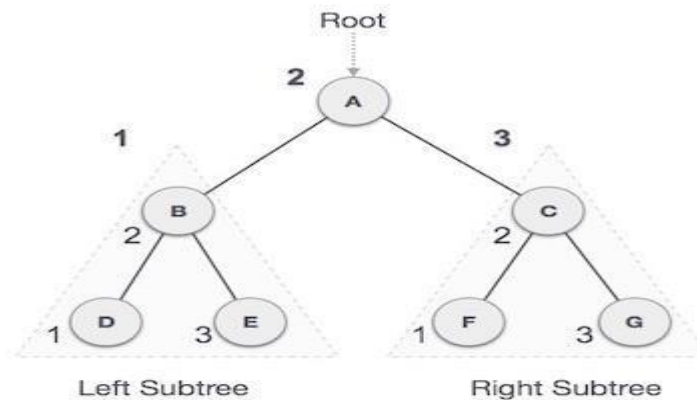


Figure 6.12: Example

Inorder: $D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$
 Preorder: $A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$
 Postorder: $D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$

2. Traverse the following binary tree into preorder, postorder and inorder.

Inorder: B D A G E C H F I.
 Preorder: A B D C E G F H I.
 Postorder: D B G E H I F C A.

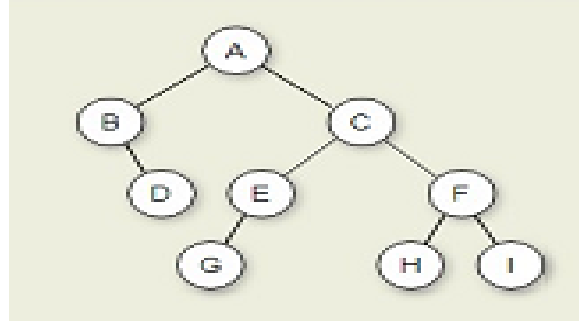


Figure 6.13: Example

3. Traverse the following binary tree into preorder, postorder and inorder.

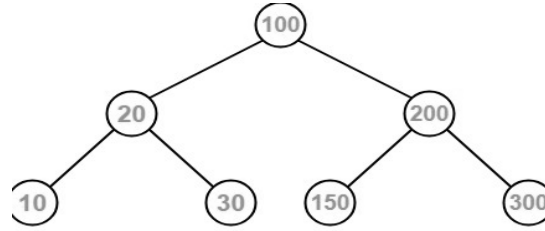


Figure 6.14: Example

Preorder: 100 , 20 , 10 , 30 , 200 , 150 , 300

Inorder: 10 , 20 , 30 , 100 , 150 , 200 , 300

Postorder: 10 , 30 , 20 , 150 , 300 , 200 , 100

4. Traverse the following binary tree into preorder, postorder and inorder.

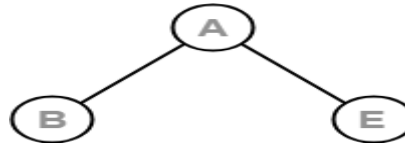


Figure 6.15: Example

Preorder Traversal : B , A , E

Inorder Traversal : B , A , E

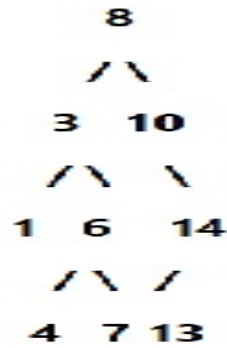
Postorder Traversal : B , E , A

6.4 बायनरी सर्च ट्री (BST) म्हणजे काय?

बायनरी सर्च ट्री (BST) ही बायनरी ट्रीची एक विशेष प्रकारची रचना आहे ज्यामध्ये प्रत्येक नोडची काही वैशिष्ट्ये असतात:

1. प्रत्येक नोडमध्ये एक की-मूल्य असते.
2. प्रत्येक नोडचे दोन सबनोड(left child आणि right child) असतात.
3. डाव्या नोडची की मूल्य सबट्रीमधील सर्व नोड्सची की मूल्य रूट नोडच्या की मूल्य पेक्षा कमी असते.
4. उजव्या नोडची सबट्रीमधील सर्व नोड्सची की मूल्य रूट नोडच्या कीपेक्षा जास्त असते.
5. प्रत्येक सबट्री स्वतः एक बायनरी सर्च ट्री आहे.

बायनरी सर्च ट्रीचे उदाहरण:



बायनरी सर्च ट्रीच्या मुलभूत ऑपरेशन्स:

1. सर्च (Search)
2. इन्सर्ट (Insert)
3. डिलीट (Delete)

1. सर्च (Search) ऑपरेशन

BST मध्ये एखादी की सर्च करण्यासाठी, रूट नोडपासून सुरू करून खालील स्टेप्स अनुसरण केल्या जातात:

- जर नोड None असेल तर की मूल्य ट्रीमध्ये नाही.
- जर नोडची की शोधत असलेल्या कीसारखी असेल तर शोध पूर्ण.
- जर शोधत असलेली की नोडच्या कीपेक्षा लहान असेल तर डाव्या सबट्रीमध्ये शोधा.
- जर शोधत असलेली की नोडच्या कीपेक्षा मोठी असेल तर उजव्या सबट्रीमध्ये शोधा.

पायथन कोड:

```

class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key
def search(root, key):
    if root is None or root.val == key:
        return root
    if key < root.val:
        return search(root.left, key)
    return search(root.right, key)
# Example usage
root = Node(8)
root.left = Node(3)
root.right = Node(10)
root.left.left = Node(1)
root.left.right = Node(6)
root.left.right.left = Node(4)
  
```

```

root.left.right.right = Node(7)
root.right.right = Node(14)
root.right.right.left = Node(13)
key = 6
found_node = search(root, key)
if found_node:
    print(f"Key {key} found in the tree.")
else:
    print(f"Key {key} not found in the tree.")

```

2. इन्सर्ट (Insert) ऑपरेशन

BST मध्ये एखादी की इन्सर्ट करण्यासाठी, खालील स्टेप्स अनुसरण केल्या जातात:

1. जर ट्री रिकामी असेल तर नवीन नोड हा रूट नोड बनवा.
2. रूटपासून सुरुवात करून, कीच्या आधारावर योग्य स्थान शोधा.
3. जर की नोडच्या कीपेक्षा कमी असेल तर डाव्या सबट्रीमध्ये जा.
4. जर की नोडच्या कीपेक्षा जास्त असेल तर उजव्या सबट्रीमध्ये जा.
5. योग्य रिकाम्या ठिकाणी नवीन नोड तयार करा.

पायथन कोड:

```

def insert(root, key):
    if root is None:
        return Node(key)
    if key < root.val:
        root.left = insert(root.left, key)
    else:
        root.right = insert(root.right, key)
    return root
# Example usage
root = insert(root, 5)
print("Key 5 inserted in the tree.")

```

डिलीट (Delete) ऑपरेशन

BST मधून एखादी की डिलीट करण्यासाठी, खालील स्टेप्स अनुसरण केल्या जातात:

1. डिलीट करण्याची नोड शोधा.
2. नोड तीन स्थितींपैकी एकात असेल:
 - a. **नोडचे कोणतेही चिल्ड्रन नाही (leaf node):** नोडला थेट काढून टाका.
 - b. **नोडचे एकच चिल्ड्रन आहे:** नोडला त्याच्या चाइल्ड ने बदलून टाका.
 - c. **नोडचे दोन चिल्ड्रन आहेत:** उजव्या सबट्रीमधील सर्वात कमी की असलेल्या नोडने (inorder successor) नोडला बदलून टाका आणि नंतर successorला काढून टाका.

पायथन कोड:

```

def min_value_node(node):

```

```

current = node
while current.left is not None:
    current = current.left
return current
def delete_node(root, key):
    if root is None:
        return root
    if key < root.val:
        root.left = delete_node(root.left, key)
    elif key > root.val:
        root.right = delete_node(root.right, key)
    else:
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left
        temp = min_value_node(root.right)
        root.val = temp.val
        root.right = delete_node(root.right, temp.val)
    return root
# Example usage
root = delete_node(root, 3)
print("Key 3 deleted from the tree.")

```

हे उदाहरण बायनरी सर्च ट्रीमध्ये शोध, इन्सर्ट आणि डिलीट ऑपरेशन्स कसे काम करतात हे दर्शविते. BST ही एक कार्यक्षम डेटा संरचना आहे जी विविध अनुप्रयोगांमध्ये वापरली जाते.

Binary Search Tree Solved Examples

1. Construct a Binary Search Tree (BST) for the following sequence of numbers-
50, 70, 60, 20, 90, 10, 40, 100

When elements are given in a sequence,

- Always consider the first element as the root node.
- Consider the given elements and insert them in the BST one by one.

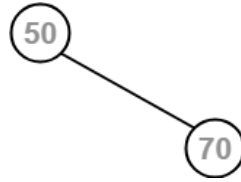
The binary search tree will be constructed as explained below-

Insert 50-



Insert 70-

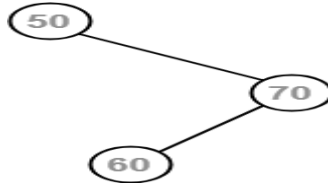
As $70 > 50$, so insert 70 to the right of 50.



Insert 60-

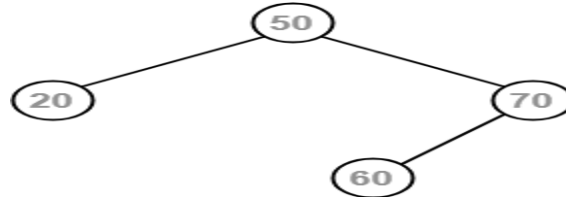
As $60 > 50$, so insert 60 to the right of 50.

- As $60 < 70$, so insert 60 to the left of 70.



Insert 20-

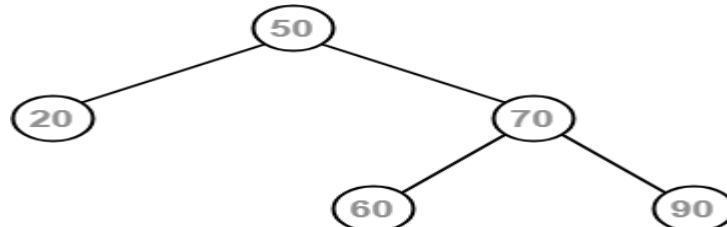
As $20 < 50$, so insert 20 to the left of 50.



Insert 90-

As $90 > 50$, so insert 90 to the right of 50.

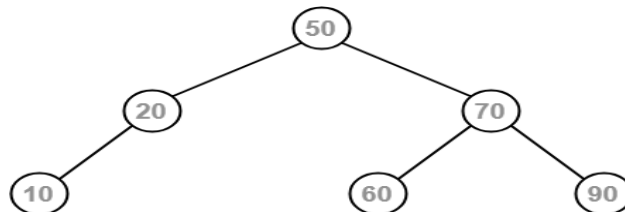
- As $90 > 70$, so insert 90 to the right of 70.



Insert 10-

As $10 < 50$, so insert 10 to the left of 50.

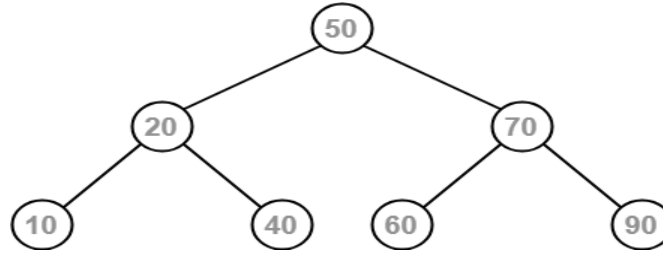
- As $10 < 20$, so insert 10 to the left of 20.



Insert 40-

As $40 < 50$, so insert 40 to the left of 50.

- As $40 > 20$, so insert 40 to the right of 20.



Insert 100-

As $100 > 50$, so insert 100 to the right of 50.

- As $100 > 70$, so insert 100 to the right of 70.
- As $100 > 90$, so insert 100 to the right of 90.

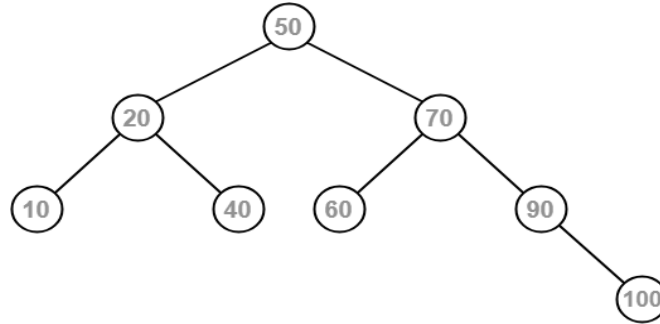


Figure 6.16: Example

This is the required Binary Search Tree.

6.5 ट्री डेटा स्ट्रक्चर विविध अनुप्रयोगांमध्ये वापरला जातो. ट्री स्ट्रक्चरची काही प्रमुख अनुप्रयोगांची यादी खाली दिली आहे: (Use of Tree data Structure)

1. फाइल सिस्टम (File System)

- फाइल सिस्टममध्ये ट्री डेटा स्ट्रक्चरचा वापर करून फाइल्स आणि फोल्डर्सची हायरेरकील स्ट्रक्चरमध्ये व्यवस्था केली जाते.

2. डेटाबेस इंडेक्सिंग (Database Indexing)

- डेटाबेसमध्ये डेटा इंडेक्स करण्यासाठी B-Tree आणि B+Tree सारख्या ट्री डेटा स्ट्रक्चरचा वापर केला जातो. यामुळे क्वेरी कार्यक्षमता वाढते.

3. कम्पाइलर आणि पार्सर (Compiler and Parser)

- कम्पाइलर्समधील सिंटॅक्स ट्री आणि ॲब्सट्रॅक्ट सिंटॅक्स ट्री (AST) हे ट्री डेटा स्ट्रक्चरच्या स्वरूपात असतात जे सोर्स कोडचे विश्लेषण करतात.

4. राऊटिंग अल्गोरिदम्स (Routing Algorithm)

- नेटवर्क राऊटिंग अल्गोरिदम्समध्ये ट्री स्ट्रक्चरचा वापर करून रूट्स आणि सबनेट्सची व्यवस्था केली जाते.

5. आर्टिफिशियल इंटेलिजेंस (AI) आणि मशीन लर्निंग (ML)

- AI आणि ML मधील निर्णय ट्री (Decision Trees) हा एक लोकप्रिय डेटा स्ट्रक्चर आहे ज्याचा वापर वर्गीकरण आणि भविष्यवाणी करण्यासाठी केला जातो.

6. हफमन एनकोडिंग (Huffman Encoding)

- हफमन एनकोडिंगमध्ये हफमन ट्री डेटा स्ट्रक्चरचा वापर करून डेटा संकुचन साध्य केले जाते.

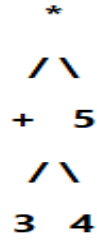
7. सोशल नेटवर्क्स (Social Network)

- सोशल नेटवर्किंग साइट्समध्ये यूजर्सच्या संबंधांची ट्री स्ट्रक्चर ठेवली जाते.

6.5.1 एक्सप्रेशन ट्री म्हणजे काय?

एक्सप्रेशन ट्री (Expression Tree) हा डेटा स्ट्रक्चर आहे जो गणितीय एक्सप्रेशन चे प्रतिनिधित्व करतो. एक्सप्रेशन ट्रीच्या रूट जवळ एक ऑपरेटर असतो आणि प्रत्येक आंतरिक नोड (internal node) हा ऑपरेटर असतो. लिफ नोड्स (leaf nodes) हे ऑपरेटरच्या ऑपेरान्ड्स (operands) असतात.

उदाहरणार्थ, $(3 + 4) * 5$ या व्य एक्सप्रेशन साठी एक्सप्रेशन ट्री असा असेल:



वरील एक्सप्रेशन ट्रीमध्ये, * हा रूटनोड आहे. * च्या डाव्या बाजूला + ऑपरेटर आहे, ज्याचे ऑपेरान्ड्स 3 आणि 4 आहेत. * च्या उजव्या बाजूला 5 आहे.

व्यक्तीकरण वृक्षाच्या मुख्य प्रकारच्या ट्रावर्सल्स (traversals) खालीलप्रमाणे आहेत:

1. **पूर्वक्रम (Preorder) ट्रावर्सल:** रूट -> लेफ्ट नोड -> राईट नोड
2. **मध्यक्रम (Inorder) ट्रावर्सल:** लेफ्ट नोड -> रूट -> राईट नोड
3. **उत्तरक्रम (Postorder) ट्रावर्सल:** लेफ्ट नोड -> राईट नोड -> रूट

वरील व्यक्तीकरण वृक्षाचे ट्रावर्सल्स खालीलप्रमाणे असतील:

- पूर्वक्रम ट्रावर्सल: * + 3 4 5
- मध्यक्रम ट्रावर्सल: 3 + 4 * 5
- उत्तरक्रम ट्रावर्सल: 3 4 + 5 *

व्यक्तीकरण ट्री विविध गणितीय आणि प्रोग्रामिंग समस्यांमध्ये सबयुक्त आहे, जसे की व्यक्तीकरणाचे मूल्यमापन (evaluation) आणि कोड जनरेशन (code generation).

डेटा स्ट्रक्चरमध्ये एक्सप्रेशन ट्रीचे गुणधर्म

- ऑपरेंड नेहमी लीफ नोड्सद्वारे दर्शविले जातात. हे ऑपरेंड नेहमी ऑपरेटर्समध्ये वापरले जातात.
- प्रिफिक्स एक्सप्रेशन, पोस्टफिक्स एक्सप्रेशन आणि इनफिक्स एक्सप्रेशनचे मूल्यांकन करण्यासाठी एक्सप्रेशन ट्री ट्रॅव्हर्स केली जाऊ शकते.

अल्गोरिदम एक्सप्रेशन ट्री तयार करण्यासाठी अल्गोरिदम

1. एक्सप्रेशन डावीकडून उजवीकडे स्कॅन करा. (Scan the expression from left to right.)
2. ऑपरेंड आढळल्यास, नंतर या ऑपरेंडसाठी एक नोड तयार करा आणि त्यास स्टॅकमध्ये ढकलून द्या. (If an operand is found. Then create a node for this operand and push it into the stack.)
3. ऑपरेटर आढळल्यास, स्टॅकमधून दोन नोड्स पॉप करा आणि ऑपरेटरला रूट नोड म्हणून आणि दोन आयटम डाव्या आणि उजव्या बाजूला ठेवून एक नोड बनवा. स्टॅकमध्ये नवीन नोड पुश करा.

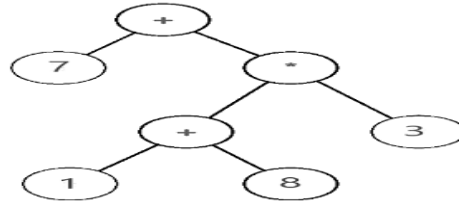
(If an operator is found. Pop two nodes from the stack and make a node keeping the operator as the root node and the two items as the left and the right child. Push the newly generated node into the stack.)

4. आम्ही आमच्या पोस्टफिक्स अभिव्यक्तीच्या शेवटी पोहोचेपर्यंत वरील दोन चरणांची पुनरावृत्ती करत रहा. (Keep repeating the above two steps until we reach the end of our postfix expression.)

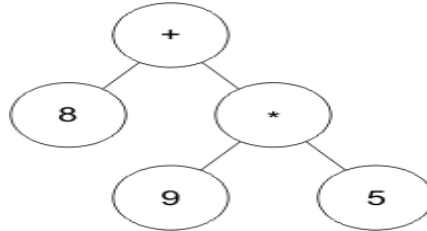
5. स्टॅकमध्ये मागिल नोड एक्सप्रेशन ट्रीच्या रूटचे प्रतिनिधित्व करतो. (The node that is left behind in the stack represents the head of the expression tree.)

उदाहरणे

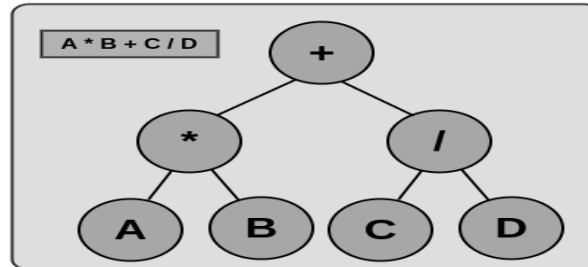
1. Draw the expression tree of given expression $7 + ((1+8)*3)$



2. Draw the expression tree of given expression $8 + (9 * 5)$



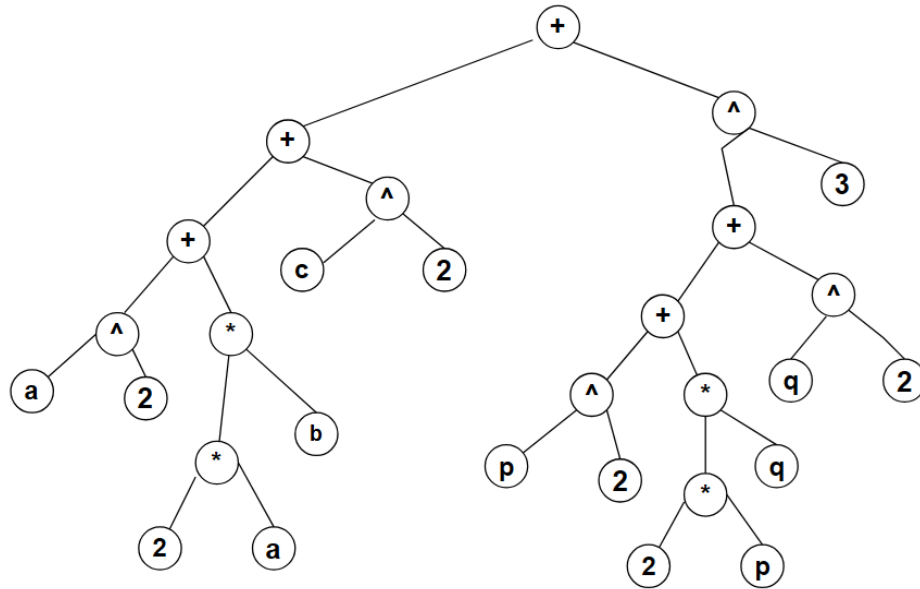
3. Draw the expression tree of given expression $A * B + C / D$



4. Draw the expression tree for $(a^2+2ab+c^2) + (p^2+2pq+r^2)^3$

Let's rewrite expression with operator as

$$(a^2+2*a*b+c^2)+(p^2+2*p*q+r^2)^3$$



Write a python program to generate binary tree

class Node:

```
def __init__(self, value):
    self.value = value
    self.left = None
    self.right = None
```

class BinaryTree:

```
def __init__(self):
    self.root = None

def insert(self, value):
    if self.root is None:
        self.root = Node(value)
    else:
        self._insert(self.root, value)

def _insert(self, current_node, value):
    if value < current_node.value:
        if current_node.left is None:
            current_node.left = Node(value)
        else:
            self._insert(current_node.left, value)
    elif value > current_node.value:
        if current_node.right is None:
            current_node.right = Node(value)
        else:
            self._insert(current_node.right, value)
    else:
        pass
```

```
        print(f"Value {value} already exists in the tree.")
def inorder_traversal(self, node, visit):
    if node is not None:
        self.inorder_traversal(node.left, visit)
        visit(node.value)
        self.inorder_traversal(node.right, visit)
def preorder_traversal(self, node, visit):
    if node is not None:
        visit(node.value)
        self.preorder_traversal(node.left, visit)
        self.preorder_traversal(node.right, visit)
def postorder_traversal(self, node, visit):
    if node is not None:
        self.postorder_traversal(node.left, visit)
        self.postorder_traversal(node.right, visit)
        visit(node.value)
def print_value(value):
    print(value, end=' ')
# Example usage
bt = BinaryTree()
values = [7, 4, 9, 1, 6, 8, 10]
for val in values:
    bt.insert(val)
print("Inorder Traversal: ", end="")
bt.inorder_traversal(bt.root, print_value)
print()
print("Preorder Traversal: ", end="")
bt.preorder_traversal(bt.root, print_value)
print()
print("Postorder Traversal: ", end="")
bt.postorder_traversal(bt.root, print_value)
print()
```

Output:**Here's the complete output:**

Inorder Traversal: 1 4 6 7 8 9 10

Preorder Traversal: 7 4 1 6 9 8 10

Postorder Traversal: 1 6 4 8 10 9 7

संदर्भ सूची (References) :

1. <https://www.codecademy.com/learn/learn-data-structures-and-algorithms-with-python/modules/trees/cheatsheet>
2. https://www.tutorialspoint.com/python_data_structure/python_binary_tree.htm
3. https://youtu.be/-xJvpnenx6Y?si=D1_HbreWMnWzXv55
4. <https://www.pythonforbeginners.com/data-structures/tree-data-structure-in-python>
5. <https://www.programiz.com/dsa/trees>