

1/July/2020

Python (27 Feb 1991)

Version 1 (1994)

- Python Basics
- GUI app with Python
- Web app with Python
- Python for data Science
- Python for AI & Machine Learning
- Bigdata Analytics with Python
- Python for IOT
- Developer of Python
Guido van Rossum

Print("Hello")

Print("Hello\n Students")

Hello

Students

Comments in Python

Single line comment

''' ''' Multiline

Variables: Reserved memory locations to store values

- Variables are dynamically typed.

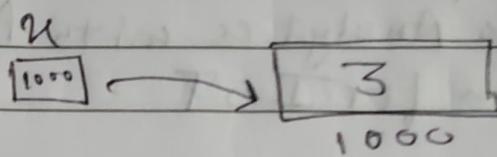
Program? → Data Processing.

Identifier: The smallest identifying unit in the program called identifier

- An identifier can denote various entities like variable, type tables, subordinate function and so on

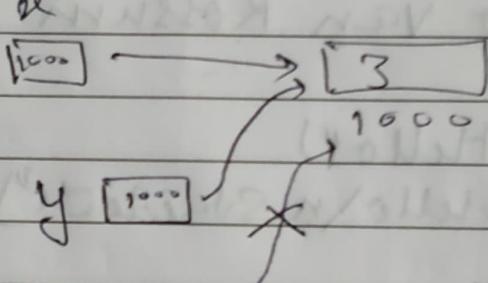
Memory concept

$x = 3$

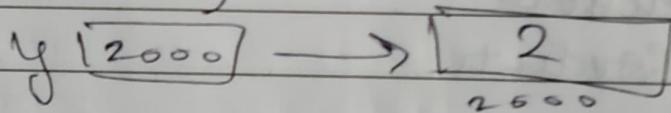


Block x refers to the actual block

$y = x$



$y = 2$



$x = 3$

Print (id(x))

$x = 3\cdot7$

Print (type(x))

- Every variable in Python is an object

Keywords in Python

- Predefined words or reserved words
- They cannot be used as variable names as their meaning is already reserved.
- There are 33 keywords in Python 3
- `import` Keyword
`Print [Keyword, Kw18t]`

Data types in Python

- Data type is category of data.

Program? → ~~A code to~~ Process data

① memory

$$a=4$$

$$a \rightarrow [4]$$

② operation

$$1.0$$

③ Representation

Dynamically typed.

In Python there are 14 built-in data type

• int $a=4$

• set

• float $a=3.7$

• frozenset

• complex $3+4j$

• dict

• bool $u=True, v=False$

• NoneType

• str $u="Amit's"$

• bytes

• bytearray

Sequence types

• range

• list

• tuple

type conversion in Python

Conversion function

- int() # returns int
- float() # returns float
- ord() # returns unicode
- str()
- complex()

$n = "123"> y = 5 > n+y \times > \text{int}(n) + y \checkmark$

- dict()
- tuple()
- list()
- set()

Handling Number System in Python

Number System

Decimal

0 to 9

(572)₁₀

Binary

0, 1

(10100)₂

Octal

0 to 7

(432)₈

Hexadecimal

0 to 9

(3f)₁₆

$n = 45$

$n = 0b45$, $n = 0B45$ } Binary

$n = 0_45$, $n = 00\ 45$ } Octal

$n = 0x3f$, $n = 0X3f$ } Hexa

Print (hex(n)) , (oct(n)), (bin(n))

Case sensitive

Slicing $\underline{0\ b\ 1\ 0\ 1}$ $n = 0b101$

$y = bin(n)$

Print (y[2:1])

#

Operators in Python

- Arithmetic operator

/ → Always float

// → Floor value.

$$-4 // 3 \Rightarrow 2 \quad (\text{So denominator ke sign mei vahi aayega})$$

Order of Precedence

** > /, //, *, %. > + -

Divide Ki Value always float

+, - → works on both (no. or string)

* → when use with string, repetition
"ab" * 3 = 'ab ab ab'

- Relational operators

True → 1 False 0

Bool(1) Bool(0)

u = True

y = int(u)

> greater than

< less than

>= greater equal

<= less than equal

== equal to { equality operation work }

!= Not equal to { with complex no. } → Not give errors

Boolean fun works

True + 5 = 6

False + 8 = 8

'a' == 97 False.

ord(a) == 97 // convert in unicode.
True.

3 == "three" → False , 3 == "3" → False
== checks content and type
== operator

int & float can be compared

Bool & int can be compared

True == 1 ✓

True == 1

True == 5 ✗

False == 6

True > 5 ✗
→ False

True != False ✓

- Logical operators

C/C++/Java

and, or → Binary operator

& & || !

not → unary operator

Python

and or not

x and y

T and T → True

T and F → False

F and T → False

F and F → False

x or y

T or T → True

T or F → True

F or T → True

F or F → False

① If x and y are non boolean then
result is also non boolean
(and)

Imp 11 if x is false then result is x otherwise
 $5 \text{ and } 4 \rightarrow 4$
 $T, \text{ then } 4 \rightarrow 4$

② (OR)

if x is false then result is y otherwise x
 $3 \text{ or } 4 \rightarrow 3$
 $0 \text{ or } 5 \rightarrow 5$

(Not) $\neg x$ $\text{Not } T \rightarrow \text{false}$
 $\text{not } f \rightarrow \text{True}$

		Not
F		T
T		F

* Str

empty "" $\rightarrow \text{false}$
"Prae" $\rightarrow \text{True}$

Error

$3 \text{ and } n=4$

No error

$3 \text{ and } 4 > 2$

$3 \text{ and } 3+4$

Bitwise operators : works on Bits (0, 1)

& and

or

~ not

^ xor

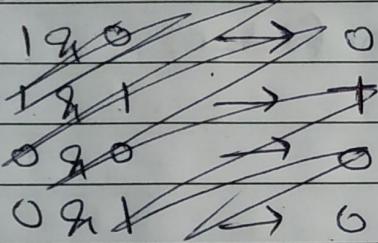
>> right shift

<< left shift

in (and)

+ for

0



(and)

1	&	1	\rightarrow	1
1	&	0	\rightarrow	0
0	&	1	\rightarrow	0
0	&	0	\rightarrow	0

(OR)

1	+	0	\rightarrow	1
1	+	1	\rightarrow	1
0	+	1	\rightarrow	1
0	+	0	\rightarrow	0

$$5 \text{ } \& \text{ } 6 = 4$$

$\begin{array}{r} 101 \\ 110 \\ \hline 100 \end{array} \rightarrow 4$

$$5 \text{ } | \text{ } 6 = 7$$

$\begin{array}{r} 101 \\ 110 \\ \hline 111 \end{array} \rightarrow 7$

$$\begin{array}{r} 101 \\ 110 \\ \hline 100 \end{array} \rightarrow 4$$

$$\begin{array}{r} 101 \\ 110 \\ \hline 111 \end{array} \rightarrow 7$$

~~64 32 16 8 6 4 2 1~~

- $(XOR) \wedge$

$$5 \wedge 6 = 3$$

$\begin{array}{r} 1 \\ \underline{+} \\ 1 \end{array}$	$\begin{array}{r} 1 \\ \underline{+} \\ 0 \end{array}$	$1 \quad 1 \quad 1 \rightarrow 0$
$\begin{array}{r} 1 \\ \underline{+} \\ 0 \end{array}$	$\begin{array}{r} 1 \\ \underline{+} \\ 1 \end{array}$	$1 \wedge 0 \rightarrow 1$
$\begin{array}{r} 0 \\ \underline{+} \\ 1 \end{array}$	$\begin{array}{r} 1 \\ \underline{+} \\ 1 \end{array}$	$0 \wedge 1 \rightarrow 1$
$\begin{array}{r} 0 \\ \underline{+} \\ 0 \end{array}$	$\begin{array}{r} 0 \\ \underline{+} \\ 0 \end{array}$	$0 \wedge 0 \rightarrow 0$

$$\begin{array}{r} 101 \\ 110 \\ \hline 011 \end{array} \rightarrow 3$$

- $\sim NOT$

$$\sim 5 \Rightarrow \cancel{101} \rightarrow -b$$

$$\begin{array}{r} 1 \\ \cancel{0} \\ 1 \\ 0 \\ 1 \end{array}$$

$$1010 \Rightarrow -K \Rightarrow K?$$

0 +ve
-1 -ve

$$\begin{array}{r} 1010 \\ \cancel{0} \cancel{1} \cancel{0} \cancel{1} \\ \cancel{+} \cancel{1} \\ \cancel{+} \cancel{1} \\ \hline 0110 \end{array} \Rightarrow b = K$$

- Right Shift ($>>$)

$$25 >> 2$$

$$25 \Rightarrow 11001$$

$$\begin{array}{ccccccc} 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \cancel{1} & \cancel{2} & \cancel{3} & \cancel{2} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{1} \\ 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}$$

$$\begin{array}{ccccccc} 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \cancel{1} & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{1} \\ 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}$$

$$25 >> 2$$

$$\begin{array}{r} 11001 \\ 01000 \\ \hline \rightarrow 00110 \end{array} \rightarrow 6$$

$$\begin{array}{r} 11001 \\ 01100 \\ \hline 00110 \end{array} = 6$$

Left Shift (LL)

12 $\ll 3 \rightarrow 96$

12 $\rightarrow 1100$

left

left me right
not hai

1100
11000
110000
 $\frac{1100000}{64\ 32} \rightarrow 96 =$

Assignment operators

=
+=, -=, *=, /=, *= } compound assignment
operator

Identity operators

- is
- is not

; Every variable in Python is an object
; objects are dynamically created

; object do not have names but reference.

x=5

y=5

x=y $\rightarrow T$

x is y $\rightarrow T$

x is not y $\rightarrow F$

x=5

x [1000] \rightarrow 5

x [1000] \rightarrow 5

y [2000] \rightarrow 6
y=6
y [2000] \rightarrow 6

10011

00110

01100

10011
00110
20006

10011
00110
20006

membership operators

- in
- not in

in: To check find search data

data in n

↑ collection/
iterable

n = "abc"

"a" in ~~n~~ n → True

error case.

n = 256

s in n → error

n = "256"
"s" in n → T

works for string

9/8/20

Python

Taking input from user

```
x = input("Enter your name")
```

input function return string.

input function/method return value as a string

So we have to convert it in int, float, complex
to perform mathematical operations.

```
y = int(x)
```

```
print(y)
```

```
Enter x = input("Enter no")
```

```
x = Print(x)
```

```
y = int(x)
```

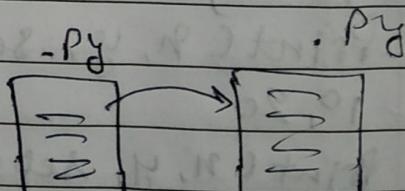
```
Print(*type(x), type(y))
```

Use of import, as and from

Module: Module is a Python file consisting of Python code/script.

Module can define functions, classes and variable.

```
import Math
math.factorial(5)
math.sqrt(5).
```



```
import Sys    ] to check import dir. Path
Print(Sys.Path)
```

```
Sys.Path.append('dirname') //append dir path
↓      ↓
var   bin
```

- Module alias,

↳ alternative name.

import Math as m

m.sqrt()

m.factorial()

- from math import factorial, sqrt.

↓

To access factorial directly

from math import *

↓

import all math functions

gcd(4, 6) → 2.

#

More on output Statement

Sep = ", " ⚡ separator

x = 10

abc + def

y = 20

abc def

Print(x, y)

10 20

By default space
is a separator

Print(x, ", ", y)

10 , 20

Print(x, y, Sep = ", ")

10, 20

Print(x, y, Sep = ":")

10:20

Print(x, y, Sep = " \n")

10

20

Print is similar as println (By default change the line)

Print("Hello", end=".")

end = "",

Print("Students")

replace newline

Print("Learn Python")

with space

x=10

y=20

z=30

Print(x,y,z, sep=":", end=".")

- Formatted String

x=10

Print("Value of x is %.d" % x)

Nahitagna

a=3

b=4

c=a+b

Print("Sum of %.d and %.d is %.d" % (a,b,c))

x=10.

y=3.5

Print("x=%.d, y=%.f" % (x,y))

x=10.1

y=10.1

Replacement operator { }

- Pair of curly braces is known as replacement operator.

$n = 10$

$y = 3.5$

$z = "Amit"$

`Print("Hello {2}, n={0}, y={1}").format(n,y,z)`

`Print("Hello {1}, x={0}, y={2}").format(n,y,z)`

`Print("Hello {2}, n={1}, y={0}").format(n,y,z)`

Decision control Statement

- Flow control Statements
- Decision control Statements

Flow control Statements

- Decision / Selection Control Statement
 - if
 - if else
 - if elif else
- Iterative / Loop Statement
 - while
 - for

if condition :

} indentation

else :

• if elif else

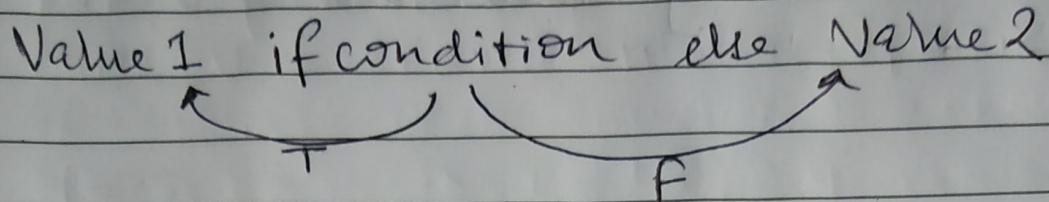
if condition :

elif condition :

elif condition :

else : } optional

Ternary Operator / Single line if expression /
conditional operator



value of
conditional operators can be in a variable.

~~$z = "Hello"$~~

$z = \text{"leap year"} \text{ if } n \cdot 400 == 0 \text{ else } \text{"leap year"} \text{ if } n \cdot 100 == 0 \text{ and } n \cdot 4 != 0 \text{ else } \text{"Non leap year"}$
 $n \cdot 100 != 0$

Iterative Statement - while loop

While loop is used when there is a condition

for loop is used when there is a sequence

While condition :

14/8/20

Iterative Statement - for loop

- While
 - Iterate statements till some condition is true
- for
 - Iterate statements till for number of element in a sequence
 - Number of iteration is known in advance

C/C++/Java

```
for(i=1; i<=10; i++)  
{
```

```
}
```

Python

for element/variable in sequence:

Sequence of elements : eg - Str, List, tuple, range, dict, set.

$n = "My String"$

for e in n:

```
=====
```

$e = M, n = M, 1$

$e = y, n = y, 2$

$e = s, n = 3$

$e = i, n = 4$

iterative control - range() function

range function generates sequence of int values and only accepts int value.

range(5) → 0, 1, 2, 3, 4

① Print(range(5))
→ range(0, 5)

② for x in range(0, 5)

Print(x)

→ 0
1
2
3
4

③ range(n) 0 to n-1

4) range (start, stop, step)

↓
By default
0

↓
By default
1

Increment by 1

e.g. range(0, 5, 2)

→ 0
→ 2
→ 4

⑤ To Print the result of for loop we use range function
for loop.

Transfer statement in Python

break, continue and Pass → transfer statement

- Ctr + C to stop infinite loop
- Break: The loop is immediately terminated and the program control resumes at the next statement
- Continue: we can stop the current iteration and continue with the next. Skips the iteration.
- Pass: Pass keyword is used to create empty block

Use of else with loop

- 1) if else
- 2) if elif else
- 3) Single line if else (Ternary operator)
- 4) while else
- 5) for else

While loop terminates in two ways.

- 1) Break
- 2) condition false,

else block can else block compiled
not compiled.

- When break is executed else will not work
- else work when condition of while is false
- In case of for loop else will execute when normally life of loop is complete,

16 | Aug/20

Sequences

List in Python (Part - 1)

List

- Lists are very similar to arrays
- They can contain any type of variable, they can contain as many variables as you wish
- List can store heterogeneous type of data
- Lists are iterable (for loop)

L = [] // empty list

L.append(10)

Print(L) Print(L[0]) note - Range function is used to generate sequence

[10]

for i in L:

Print(i)

List in Python (Part 2)

Create List

1) L = []

2) L1 = [10, 20, 30]

3) L2 = ["Bhopal", "Delhi", "Jaipur"]

4) L3 = [10, 3.5, 30, "Jaipur"]

5) exception

L = [10, 20, 30]
[0] [1] [2]

Print(L[3]) → error, index out of range

6) L2 = []

[0] index does not exist,

L2[0] = 10

value can't be inserted

error, index out of range

• $l2 = [10, 20]$

$\rightarrow [10, 20]$

$l2 = [1, 2, 3]$ // new values of list replace

$\rightarrow [1, 2, 3]$ the previous values.

• `append()`

add value at the end of the list

$a = []$

a. `append(10)`

a. `append(20)`

$\rightarrow [10, 20]$

• `insert()`

In insert method we can specify the index and object

L. `insert(index, object)`

$a = [10, 20]$

a. `insert(2, 30)`

$\rightarrow [10, 20, 30]$

a. `insert(4, 40)`

$\rightarrow [10, 20, 30, 40]$ // insert value at last
0 1 2 3 possible index

a. `insert(1, 50)` // to insert at index [1]

$\rightarrow [10, 50, 20, 30, 40]$ and shift other
0 1 2 3 4 values by 1 index

a. `insert(-1, 60)` // for -ve indexing count

$\rightarrow [10, 50, 20, 30, 50, 40]$ from right to left

-5 -4 -3 -2 -1 0

a. `insert(-3, 100)` // to 20, left shift

$\rightarrow [10, 50, 20, 100, 30, 60, 40]$

-6 -5 -4 -3 -2 -1 0

- $A = [10, 20, 30]$
 $B = [40, 50, 60]$
 $C = A + B$
 $\rightarrow [10, 20, 30, 40, 50, 60]$
- $A + [10]$
 $\rightarrow [10, 20, 30, 10]$ // Here 10 is not stored in list
only prints
- $A += [10]$ // to insert / add value
 $A = A + [10]$ Append alternate meth.
 $\rightarrow [10, 20, 30, 10]$
- $A += 10$ // Error, int type
can't be
append
- $D = [1, 2, 3]$
 $D *= 3$
 $\rightarrow [1, 2, 3, 1, 2, 3, 1, 2, 3]$ • only iterable seq. can append
- ~~$D *= 3$~~ // to change in D use
compound assignment
 ~~$D = D * 3$~~ operator

list in Python (Part 3)

- To modify element in list
 $a = [10, 20, 30]$
 $a[0] = 100$
 $\rightarrow [100, 20, 30]$
- To remove element from list
 $a = [10, 20, 30]$
 $a.remove(20)$ // Pass value (20)
 $\rightarrow [10, 30]$

- $D = [10, 20, 10, 20, 10, 20]$
 $D.\text{remove}(20)$ // first occurrence
 $\rightarrow [10, 10, 20, 10, 20]$

- List also is an object
Sorting $\rightarrow \text{Sort}()$
 $D.\text{Sort}()$
 $\rightarrow [10, 10, 10, 20, 20]$

- To remove all values from list

$D.\text{clear}()$

$\rightarrow []$

$D = [1, 2, 3, 4]$

→ None By Default

$\text{Sorted}(l, \text{Key}, \text{reverse=False})$

optional

- To reverse values of list

$D.\text{reverse}()$

$\rightarrow [4, 3, 2, 1]$

$l.\text{Sort}(\text{Key}, \text{reverse=False})$

- $\text{Pop}()$

extracts the value from list. By default

extracts last element.

and return the value ~~pop~~ extracted/Poped value.

~~#~~. $D.\text{Pop}()$

$\rightarrow 1$

$D = [4, 3, 2, 1]$

$D = [4, 3, 2]$

$D.\text{Pop}(0)$

$\rightarrow 4$

$D = [3, 2]$

$x = D.\text{Pop}(1)$

$x = 2$

$D = [3]$

l. Sort (Key = lambda x : x.name , reverse = True)

A = [10, 30, 30, 40]

A. remove (20)

A

→ [10, 30, 40]

A. pop (2)

→ 30

A

→ [10, 30]

- index () // To determine the index of the particular value

A = [10, 20, 30, 40]

A. index (30)

→ 2

A = [10, 20, 10, 30, 40, 20, 30, 10]

A. index (10, 1, 6)

↓ ↓ ↓
element start end
 index index

A. index (10, 1) // we can pass upto

→ 2 ~~or three arguments~~

- count () // To count the occurrence of a Particular element.

A. count (10)

→ 3

- help (A) // A = list variable

~~help (A)~~

Command line Arguments

run Python script through command prompt.
we can pass argument through command line.

Python Hello.py Rhopal indore Delhi
→ 'Hello.py', 'Rhopal', 'indore', 'Delhi'
accept data as string.

- From sys import argv
Python Hello.py A B C

From sys import argv

for u in argv:

print(argv[u], type(u))

From sys import argv

y=0

s=0

for x in argv:

if y==0:

y=1

else:

s=s+int(u)

Print("Sum is", s)

- Array input in Python.

arr= map(int, input().split())

Strings in Python Part-1

Strings can be written in 3 ways

- Single quote 'Praveen'
- double quote " Praveen"
- triple quote """ Praveen """

* In triple quote we can pass
multiline strings.

len(s1) → to calc. length of string

To find index. | To count occurrence
s1.index('s') | s1.count('e')

Strings in Python Part-2

s1 → mySirG education Services

Negative indexing.
→ Right to left.

mySirG education Services
-87654321

* Using Operator

[beg : end : step]
↑ ↑ ↗
include exclude +1
OPTIONAL

`SI[7:16]`

→ 'education'

`SI[-5:8:-1]`

' vres noitacu'

To Reverse a string. By using slicing

`SI[24::-1]`

at zeroth index, m
~~can't~~ not prints.

`SI[len(SI):-1]`

`SI.upper()` → Converts String in uppercase

`SI.lower()` " " " lowercase

`SI.startswith(" my")`

True.

`SI.endswith(" ces")`

True

`Split()` → Splits the string into list of strings.

`SI.split(' ')`

→ ['mysing', 'education', 'Services']

`help(str)`

Tuple Part -1

Tuple \rightarrow Sequence of elements

Tuple () (1, 2, 3)
 ° 1 2

List is mutable. (change is allowed)

Tuple is immutable (no change)

t = () \rightarrow tuple

t = (10) \rightarrow 'int'

t = (10,) \rightarrow tuple

t = 10, 20, 30 \rightarrow tuple

l = [11, 22, 33, 44]

t = tuple(l) l is of string type

t = tuple(10, 20, 30) X

t = tuple((10, 20, 30))

t = (10, 4.5, "xyz")

Print through indexing.

t[0] \rightarrow 10

t[1] \rightarrow 4.5

Printing tuple via

1) loop

↳ while

↳ for

2) slicing operator

$t = (10, 20, 30, 40)$

$i = 0$

while $i < \text{len}(t)$:

 print(t[i])

i += 1

}

while

for x in t

 print(x)

}

$t[0 : :]$

}

Slicing

$t = t + (1, 2, 3, 4)$

$\rightarrow [10, 20, 30, 40, 1, 2, 3, 4] \quad // \text{New tuple}$

$t[2:5: :] \rightarrow 30, 40, 1$

$t[2:5] \rightarrow 30, 40, 1$

tuple can only concatenate tuple not 'int' to tuple

Tuple (Part 2)

Packing.

~~tup~~ a=1

b=2

c=3

$t = (a, b, c)$

Unpacking.

$t = (10, 20, 30)$

a, b, c = t

* Concatenation

$a = 1, 2, 3 \quad b = 10, 20$

$a + b \Rightarrow (1, 2, 3, 10, 20)$

* Repetition operator

$t = 1, 2, 3$

$t * 2$

$(1, 2, 3, 1, 2, 3)$

Comparison operators

a = 1, 2, 3

b = 1, 2, 3

c = 1, 2, 4

$a == b$	$a == c$	$a <= c$	
True	False	True	

Tuple Part 3

1) `x = tuple(input())`

2) ~~so~~ `y = eval(input())`

eval means evaluate expression

`x = tuple(input())` → 1, 2, 3

('1', '1', '2', ' ', ' ', '3',)

tuple function treat , as data and of string type so we use eval function to input data in tuple

Important functions in tuple

- | | |
|-----------------------------|--------------------------|
| 1) <code>len(u)</code> | $u = 10, 20, 10, 20, 10$ |
| 2) <code>u.count(10)</code> | |
| 3) <code>u.index(10)</code> | |
| 4) <code>max(u)</code> | |
| 5) <code>min(u)</code> | |

list

- 1) mutable
- 2) []
- 3) Not used as KEY values

Tuple

- 1) immutable
- 2) () optional
- 3) used as Key values

Set in Python Part-1

1) No duplicate values

2) $S = \{10, 20, 30\}$

3) No indexing

$S[0]$ X

4) $S[1:5:2]$ X

No. Slicing operator

5) mutable (changes are allowed)

6) Create set

$S = \{10, 20, 30\}$

$S = \{\}$ → dict type not a set

$S = \text{set}()$ || to create empty set

7) Value in set store in random order.

$S = \text{set}([10, 20, 30, 1])$

$S = \text{set}((10, 20))$

$S \rightarrow \{10, 20, 30\}$

$S = \text{set}("mydirg.com")$

$\{m, i, s, ., r, y, o, c, g\}$

$S = \text{set}()$

↑ sequence enter karne mai. (list, tuple)

(int, float) X

$l = [10, 20, 10, 30, 10, 20]$

$S = \text{set}(l)$

S

$\rightarrow \{10, 20, 30\}$

#

Set in Python Part 2

- $S = \{1, 2, 3\}$

$S.add(4)$

S

$\{1, 2, 3, 4\}$

- $S = \text{Set}(10)$

$S = \text{Set}((10, 20))$

S

$\rightarrow \{10, 20\}$

- $S = \text{Set}(3.5)$

S

$\{1, 2, 3.5, 3, 4\}$ Sequence not guaranteed

- $S = \{1, 2, 3\}$

$L = [1, 3, 5, 7]$

$S.update(L)$ // Update fun. Performs union.

$S = \{1, 2, 3, 5, 7\}$ ($S \cup L$)

- $S.update([3, 5, 9], (5, 6, 7, 8))$

list

tuple

* $S = \{1, 2, 3, 5, 6, 7, 8, 9\}$

- $S.discard()$

// no error

$S.remove()$

// error when element not found.

- $S = \{1, 2, [3, 4], 5, 6\}$
 error: unhashable type: list
 || we can't use list as an element in set
- $S = \{1, 2, (3, 4), 5, 6\}$
 s // tuple is used as element in set
- $S = \{1, 2, (3, 4), 5, 6\}$
 $S = \{1, 2, (3, 4)\}$
 $S = \{1, 2, 5, 6\}$
- $S1 = \{1, 2, 3, 5, 8\}$
 $S2 = \{2, 3, 4, 6\}$
 $S1.Sorted()$
 $S1.union(S2)$
 $S1.intersection(S2)$
 $S1.discard(3)$
 $S1.pop()$
 $S1.clear()$
 $S1.issubset(S2)$
 $S1.issubset(S2)$
 $S1.pop()$ // in sets Pop() fun. removes any1 element randomly
- $S1 = S.copy()$ // makes copy of the set
 $S1 = S$ // element of S copies in set S1
- $id(S1) \approx 4786504$, $id(S) \approx 47866864$

Dictionary Part-1

Object / value

list $\{ [10, 20, 30] \}$ individual values

Tuple $(10, 20, 30)$

Set $\{ 10, 20, 30 \}$

Imp.

Each element of dictionary is a pair of Key-Value

Key	Value
1) Key must be unique	1) Repetition Possible
2) Any type	2) Any type
3) Heterogenous	3) Heterogeneous

- indexing not Possible
- Slicing operator not work
- Cannot guarantee of order of storage of key-value in the dictionary
- mutable : change is allowed. (add, delete, edit)
- grow / shrink.

`d = {}` || To create dictionary

`d = dict()` || create dict by constructor

`d = {100: 'Rahul', 101: 'Gamar', 102: 'Ajay'}`
By constructor dict

`d = dict(one='Bhopal', two='Indore')`

$\rightarrow \{ 'one': 'Bhopal', 'two': 'Indore' \}$

(Here in `dict()` we use `=` instead of `:`)
(and Key is By default treated as string)

- To access value by in dict() method.
 $d['one']$
 $\rightarrow \text{Bhagel}$
- $d = \{1: "A", 2: "B", 3: "C", 4: "D"\}$
 To add element in d.
 $d[5] = "E"$
 $d = \{1: "A", 2: "B", 3: "C", 4: "D", 5: "E"\}$
- To edit value in dictionary
 $d[1] = "a"$
- For K in d:
 $\text{Print("Key=", K, "value=", d[K])}$

Roll no	—	Name.
100	—	Rahul
101	—	Samar
102	—	Ajay
\downarrow	—	\downarrow
Key	-	Value

Sort(): Sorted()

list ✓

Tuple ✗

Set ✗

Dict ✓

dictionary Part-2

words = {}

- To delete any pair in dict.
`del ['c']` X // error
`del [words ['c']]`
- `words.clear()` // clears all pairs in dict.
- `d = {1: 'A', 2: 'B'}`
`d1 = d.copy()` // Shallow copy

Here d1 & d contain same key value pair
But refers to different key object

`d1` → `[(1, 'A'), (1, 'B')]`
5478910

`d1` → `[(1, 'A'), (1, 'B')]`
10571911

`d is d1` // to check id's of d & d1 is same
False or not.

- `d.fromkeys()` method.

`l = [1, 2, 3]`

`d. fromkeys(l, "ABC")`

`d.` ↳ optional, By default None

`d2 = {1: 'ABC', 2: 'ABC', 3: 'ABC'}`.

this doesn't change the value in d
we can use this to store this in another
variable like (d2)

$d = \{1: 'A', 2: 'B', 3: 'C', 4: 'D'\}$

- $d.get(2)$ // no error
 $'B'$
 $d[2]$ // gives key error
 $'B'$
 $g = d.get(3)$ // to store value returned by get(3)

- $y = d.pop(2)$ // extract the value of key 2
 y
 $'ABC'$
error when key is not valid

$(2: "ABC")$
2
"AB"

dict-item
dict-key
dict-value

- $d.popitem()$ [←] No argument
Popitem() method returns ~~pop~~ tuple value (type)
 $t = d.popitem()$
 d
 $\{1: 'A', 2: 'B', 4: 'D'\}$
 $type(t)$
 $tuple$
 t
 $(4, 'D')$
 $t[0]$
 4
 $t[1]$
 $'D'$
- help(Dict)

7/9/2020

Organize Code

More on input Part-1

`x = eval(input("Statement"))`

More on input Part-2

- To take two inputs

1) `x, y = input(), input()`

2) `x, y = input("Enter two numbers").split()`

↳ `input().split()`



Returns the list split string

`d = input("Enter date").split("/")`

`d`

`['18', '7', '2020']`

`a, b, c = input("Enter date").split("/")`

`a = '18'`

`b = '7'`

`c = '2020'`

→ String type

Reading multiple values.

`a, b, c = [int(x) for x in input("Enter date").split("/")]`

`a, b, c = [1, 2, 3] // a=1, b=2, c=3 // a, b, c = (1, 2, 3)`

More on input Part-3

- Reading multiple value of different type

`a,b,c=[eval(n) for n in input().split(',')]`
this statement accepts int, float, str.

- For List []

`L=[eval(n) for n in input("Enter three values").split(',')]`
 $\rightarrow [11, 22, 13]$

- For tuple ()

`t=tuple([eval(n) for n in input().split(',')])`
 $\rightarrow (10, 20, 30)$

- For String Set {}

`S=set([eval(n) for n in input().split(',')])`
 $\rightarrow \{33, 11, 12\}$

- for dictionary

method 1

`d=dict(input().split('-')) for _ in range(3)`

`1 - 'a'`

`2 - 'b'`

`3 - 'c'`

$\rightarrow \{1: "a", 2: "b", 3: "c"\}$

this treat 1, 2, 3 as string.

method 2 for dictionary input:

```
d = dict {n: input() for n in range(1, 5)}  
→ {1: 'Ajay', 2: 'Ravi', 3: 'Rahul', 4: 'Vikas'}
```

method 3

```
d = {input("Key") : input("Value") for n in  
range(3)}
```

→ Value Amit

Key 100

→ Value Rahul

Key 101

→ Value Ajay

Key 102

d

```
→ {100: 'Amit', 101: 'Rahul', 102: 'Ajay'}
```

* d = {int(input("Key")) : input("Value") for n
in range(3)}

functions

(Part-1)

- Function is a block of code
- function has some name, but in Python we can create a function without name
- Code reusability
- Their two types of function
 - Predefined function
 - Userdefined function
- How to create a function

```
def function_name():  
    statement 1  
    statement 2  
    ...
```

- How to call a function
`function_name()`
- four ways to define function
 - 1) Take nothing, Return nothing (TNRN)
 - 2) Take Something, Return nothing (TSRN)
 - 3) Take Nothing, Return Something (TNRS)
 - 4) Take Something, Return Something (TSRS)

Part-2

1) Takes Nothing, Returns Nothing

```
def add():  
    Print ("Enter two no.")  
    a = int(input())  
    local b = int(input())  
    variable S = a+b  
    Print ("sum is", S)
```

function definition

add()
Empty \leftarrow function call

No. use of return

2) Takes something, Returns Nothing

```
def add(a,b):  
    S = a+b  
    Print ("Sum is", S)
```

add(10,20)

\rightarrow Actual arguments

3) Takes Nothing, Returns Something

```
def add():  
    Print ("Enter two number")  
    a = int(input())  
    b = int(input())  
    S = a+b
```

return S

x = add()

Print ("x =", x)

|| We can also return multiple values

4) Takes Something, Return Something

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

```
    s = a + b
```

```
    return s
```

```
n = add(10, 20)
```

```
Print("Sum is =", n)
```

Part - 3

- When function returns nothing it return None

```
def f1():
```

```
    Print("Hello")
```

```
n = f1()
```

```
Print("N, ", type(n))
```

→ Hello

→ none -<none type>

- Default Arguments

```
sum() || def sum(a=0, b=0, c=0):
```

```
sum(10) || def sum(a, b=0, c=0):
```

```
sum(10, 20) || def sum(a, b, c=0):
```

```
sum(10, 20, 30) || def sum(a, b, c):
```

* after default Argument, Non default argument is not allowed

~~(a=0, b= (a=0, b, c=0) X~~

(a, b=0, c=0) ✓

Part - 4

```
def f1(a,b):
```

f1(10, 20)

Positional argument

- Keyword argument.

```
def f1(a,b):
```

```
    print ("a=", a, "b=", b)
```

f1(10, 20)

f1(b=10, a=20)

f1(20, 10) // Swapping

f1(10, b=20) // mixture of positional arg. &

Keyword arg.

f1(a=10, 20) // error : Positional arg.
follows keyword arg.

#

* if first arg is keyword arg. then after
that after that arg. every arg. must
be keyword arg.

~~#~~ Part -5

- Variable length arguments

```
def avg(*n):  
    return (a+b)/2  
n = avg(10, 20, 30)  
Print("average is", n)  
Print(type(n)) → (10, 20, 30) tuple.
```

*n → Accepts variable length arguments
→ Is of tuple type

```
def f1(*Points, Playername):  
    Print(Playername, end=' ')  
    S = 0  
    for n in Points:  
        S = S + n  
    Print("total points = ", S)
```

```
f1("Rakesh", 10, 11, 11) X  
f1(10, 11, 12, Playername="Ajay")  
f1(11, 12, 13, Playername = "Amit")
```

- Variable length Keyword Arguments

```
def f1(**K): // **K → dict type  
    Print("Person Information")  
    for Key, Values in K.items():  
        Print(Key, "-", Values)
```

f1(name="Sameer", age=22)

f1(name="Rahul", marks=87, age=23)

f1(name="Ajay", empId=125, salary=350.00)

Recursion

Function calling itself is known as recursion

Solution of the problem is defined with the help of result of simpler version of the same problem then the solution is recursive.

def f1():

=====
f1()
=====

for eg:-

def sum(n):

=====

return n + sum(n-1)

n = 5

5 + sum(4)

(4 + sum(3))
↓

3 + sum(2)
↓

2 + sum(1)
↓

def sum(n):

Base case if [n==1]:
 return 1

else
 return [n+sum(n-1)]

Recursive
case

• Advantages

- 1) easy coding
- 2) easy to read

• Approach for recursion

- 1) Sum(n) // create fun.
- 2) n + sum(n-1) // recursive case
- 3) n == 1 // Base case.

* Every recursive problem can be solved with loops also

Lambda expression

- Anonymous function, fun. without name.
- Single line function

$r = (\lambda a, b : a+b) (5, 3)$

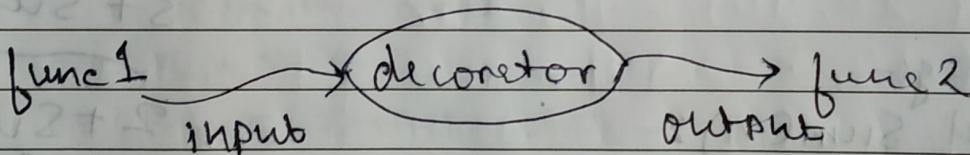
Print(r)

→ 8

- Lambda fun. always return something after colon(:)

Decorators

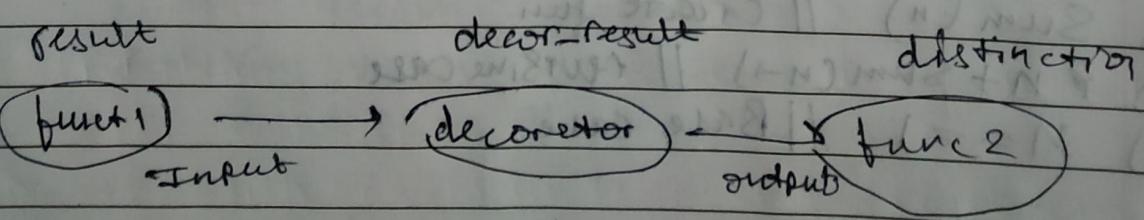
- 1) function to enhance or to decorate another fun.



$u = f_1$ → alias $| u = f_1()$

$\text{def } f_2(u) = f_2(f_1)$

- @ decorator func name to call decoration

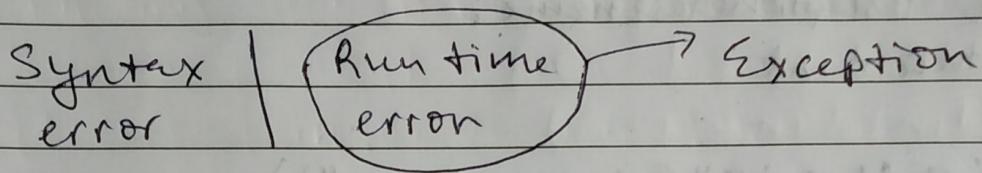


on calling result fun. distinction will execute

Exception Handling (Part -1)

• Dealing with errors

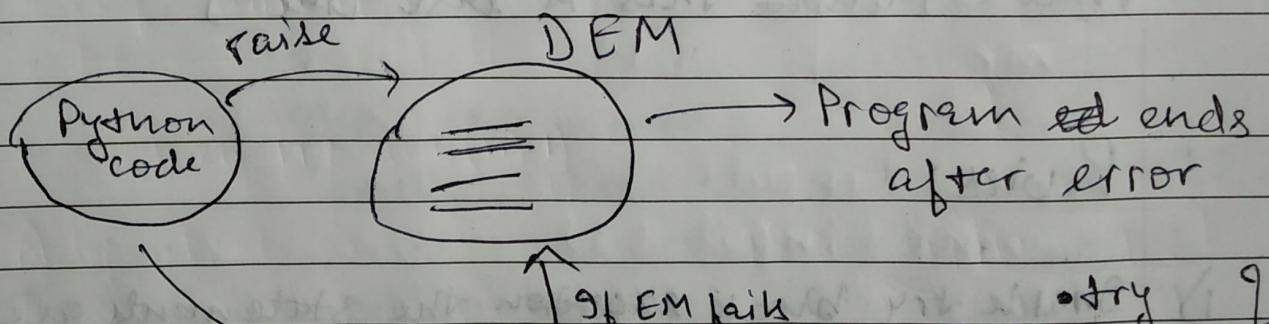
- `TypeError`: unsupported operand type(s) for +:
 - `ZeroDivisionError`: division by zero
'int' and 'str'



Type error, Zero division etc → class / type in Python called exception classes

Exception classes

Default except mechanism (DEM)



raise	Handle	
1) Python	DEM	1) Predefined exception
2) Python	EM	
3) User	DEM	2) Userdefined exception
4) User	EM	

Exception Handling Part-2

try :

 line1

 line2

 line3

except ExceptionClassName :

 Statement

x = int(input("Enter first no."))

y = int(input("Enter second no."))

try :

 z = x/y

 Print ("Division result", z)

except ZeroDivisionError :

 Print ("Invalid attempt")

 Print ("Hello this is last line")

Rules

- 1) Inside try block exception the statements are written whose probability of getting exception is max
- 2) no except Block without try block
- 3) Error name and exception class name must be same.
- 4) No exception raised in try, then except block is skipped

5) Multiple except block are allowed.

try:

except _____:

except _____:

except _____:

a = 5+'5' → typeerror

num = int("String")

→ value error

int ("5") → 5

in ("a") → value error:

invalid literal for
int()

6) If exception classname doesn't match with raised exception, the DEM (Default exception mechanism) is run.

7) Finally runs always

8) order, try

↓

except → 0 or more

↓ finally → 0 or 1

9) To handle multiple exception

except (TypeError, ValueError, ZeroDivisionError):

10) Default exception Block (other than DEM)

except :

 print("Default Exception")

11) Use of else with try, else will run when there is no exception
No exception raised then else execute

Exception Handling Part - 3

- Case 3

raised	Handle
User	DEM

raise Keyword → raise Exceptionclassname ("Msg")

x = int(input())

y = int(input())

if y == 0:

 raise ZeroDivisionError ("Denominator
 cannot be zero")

z = x/y

Print("Division is", z)

- Case 4

raised	Handle
User	EM

x = int(input())

y = int(input())

try:

 if y == 0:

 raise ZeroDivisionError ("")

 z = x/y

 Print("Division", z)

except ZeroDivisionError:

 Print("you can't divide by zero")

Userdefined Exception class

```
class InsufficientBalance (ZeroDivisionError):
    def __init__(self, arg):
        self.msg = arg
balance = 5000
w = int(input("Enter amount to withdraw"))
try:
    if w > balance:
        raise InsufficientBalance ("Insufficient
Balance in the account")
    balance = balance - w
except InsufficientBalance as i:
    print("Exception", i.msg)
else:
    print("Withdraw amount", w, "successfully")
finally:
    print("Current Balance is", balance)
```

Exception Handling - Nesting of try block (Part-4)

```
try:
    print("line 1")
    a) 3/0 b) a+15"
    print("line 2")
try:
    print("line 3")
    a) 3/0 b) 34"5"
    print("line 4")
```

except ZeroDivisionError:

 Print ("Except 1")

finally:

 Print ("finally")

Print ("line 5")

except TypeError:

 Print ("Except 2")

finally:

 Print ("finally 2")

a)

line 1

finally 2

b)

line 1

except 2

finally 2

c)

line 1

line 2

line 3

Except 1

finally 1

line 5

finally 2

d) line 1

line 2

line 3

finally 1

except 2

finally 2

Object Oriented Programming

- 1) Procedure oriented way of Programming
- 2) Object oriented way of Programming
[data + function]
→ Object
- 3) Main aspects of OOP
 - 1) Classes and objects
 - 2) Inheritance
 - 3) Polymorphism
 - 4) Data hiding
 - 5) Data abstraction

Classes and Objects Part - I

Class :

- class is a keyword
- class has some name
- class encapsulates data and functions
- creating class is creating type
- class is an description of an object
- class is an blueprint of an object
- int, float, complex, str, tuple etc are built-in class

Variable → fields } attributes
function → methods }

a.bit_length()
n.bit_length()

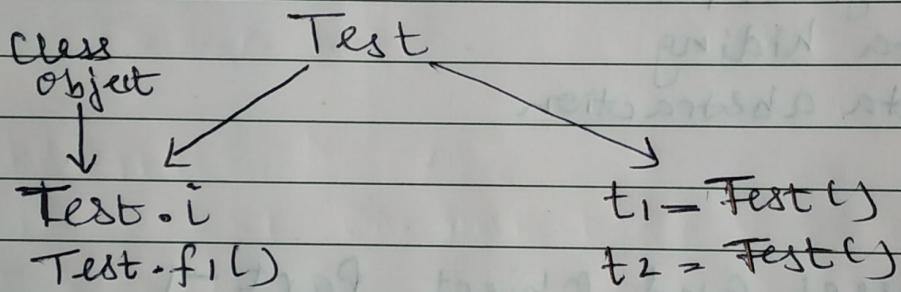
- we can create any number of objects
objects of some class, but we have to define
class first

Classes and Objects Part-2

Syntax of class

class class name : Class Test :
variable i = 10
functions def f1()

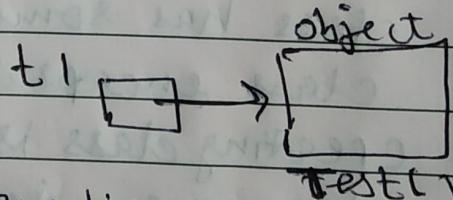
Class Test :
 $i = 10$
def f1():
 print("Hello")



Test is used as class object

Test is used as
a function

- Class object can define one time
but objects / instance can define more
than one time



Classes and Object Part - 3

Imp

`__init__()`

1) `__init__()` function automatically runs when we create object similar as constructor, and it will run on one time.

2) if we don't

`__init__()` function always take minimum one argument.

`__init__(self)`

`__init__(self, a, b):`

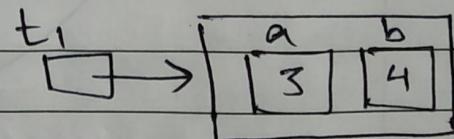
`self.a = a`

`self.b = b`

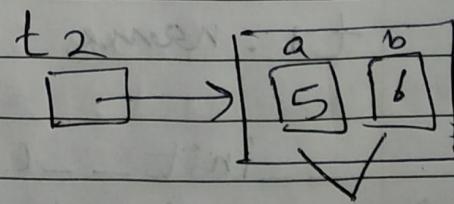
instance
variable

`t1 = Test(3, 4)`

`t1, 3, 4`



`t2 = Test(5, 6)`



Instance
variable

Classes and object Part - 4

Functions in the class

Class Test :

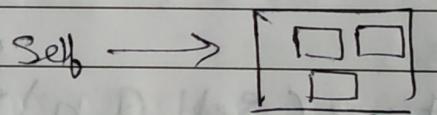
def f1(): // can't access instance variable

def f2(self):

def f2(self): access instance variable
Point(self.name)

S1.f1() ← No arg.

S1.f2() ← One implicit arg
↳ f2(S1)



def f3(x, y, z)

S1.f3(3, 4)

↳ f3(S1, 3, 4)

t1.name // to access instance variable outside class

__init__()

function

- 1) Implicit call
- 2) one time call
- 3) To declare instance variable
- 4) one arg is must

1) explicit call

2) any no. of time

3) for Problem solving logic

4) No such rule

Types of variable - instance variable

- 1) Instance variable
- 2) Static variable
- 3) Local variable
- 4) Global variable

- Instance variable

Object specific variable / Information.

- How to create instance variable

1) __init__(self)

2) methods

3) Outside the class → Object reference . instance variable

Class Account :

```
def __init__(self, a, b) { instance variable  
    self.accno = a } through init()  
    self.balance = b func.
```

```
def f1(self, a, b) : { through  
    self.accno = a } method. func.  
    self.balance = b
```

acc1 = Account()

```
acc1.accno = 102 { through outside the  
acc1.balance = 6000 } class
```

Print(acc1.__dict__) // gives details of
object as dictionary

- When we call a ~~object~~ fun from obj outside class the one arg
implicitly pass, which is current object

Types of variable & static variable

- Static variable

- 1) ~~there~~ their is not static Keyword in python
- 2) class specific (not object specific)
- 3) Shared variable among all the objects

- Creating Static variable

1) Class Test :

a = 10 # static variable

2) def __init__(self):

self.x = 1 # instance variable

Test.b = 20 # static variable

3) def f1(self): # instance member function

x = 100 # local variable

self.y = 2 # instance variable

Test.c = 30 # static variable

→ Access of S.V, this will also

4) def f2(): create S.V if not created

@ static method

def f2(m,n): # static method.

Test.d = 40 # static variable

t1 = Test()

t1.f2(3,4) # object specified calling.

obj = T1()

Point(obj, x) # accessing static variable
Instance

```
def f2(m,n):  
    print(m,n)  
Test.d=40
```

t1 = Test()

Test.f2(3,4) # calling through ~~the~~ class object

```
def f2(): # static method as no args passed.
```

5) def f3():

@ classmethod

```
def f3(cls):
```

cls.e=50 # static variable

Test.f=60 # static variable

Test.f3() # calling through class object
implicitly pass ~~the~~ class object

6) Test.g=70 # static variable outside
of class

Class method

Static Method

1) implicit arg.
class object

No implicit arg.

2) @classmethod

@ staticmethod
(optional)

can't access instance
member

Print(Test.__dict__) // gives details of
static variable as
dictionary.

Types of variable - Local and Global.

- Local variable - scope is limited to the func.

def f1(c):
 n=5 → local var.

- Global variable - declare outside of class and func.

- $y = 10$ # Global variable
we can directly use global variable
in \$ any function

- def f1():
 global y # using global Keyword, this
 y=5 also creates global variable
 Print(y)
→ 5

- $y = 10$ # Global Variable
Print(" Outside function y=", y)

def f1():
 y=5 # local variable
 Print(" Inside function local y=", y)
 Print(" Inside function Global y = ", globals()['y'])
f1()

Print(" Outside function y=", y)

→ 10

5 → Inside function Preference of local variable
is more than global var.
10
10

Inheritance Part-1

Encapsulation - Wrapping up of data member and member function into a single unit. e.g. classes and obj class.

Inheritance - Defining a new class with the help of an old class.
Reusability of code.

Type → Person (name, age) → <sup>Super
old</sup> Base Parent

Sub Type → Student (rollno, name, age) → <sup>Sub
New
Derived</sup> Child

Person (name, age)
↓ Inheritance

Student (rollno.)

- Syntax

class Derived (Baseclassname) :

e.g:-

class Person :

....

class Student (Person) :

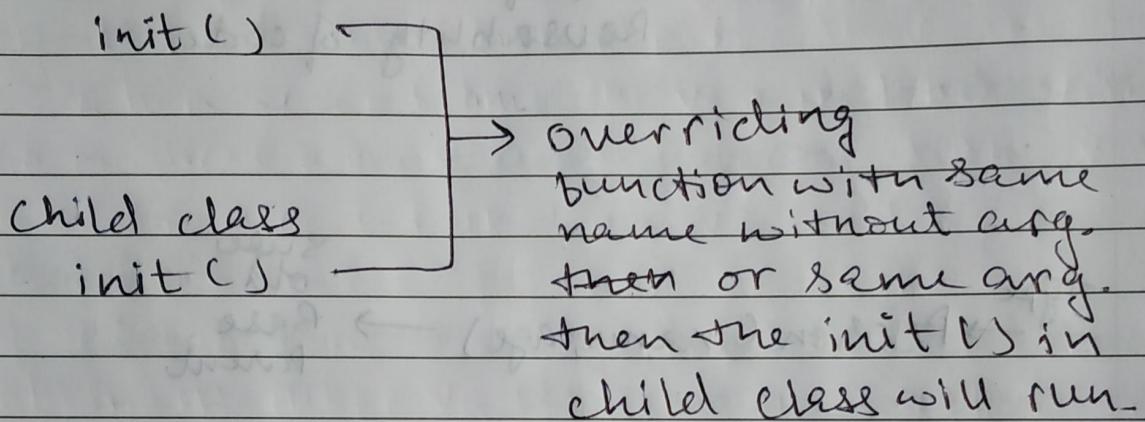
....

- ⑧ Inherit class from module
first import module then

class Derived(module.Base)

module.example()

- Parent class.



Override → If two func. with same name and arg. defined in both Parent class and child class.

then the child version of `init()` will execute. means child class `init()` overrides the Parent ~~and~~ class `init()`

Hiding → If two func. with same name but different arg. defined in both Parent class and child class.

then the child version of `init()` will execute.

The child version hides the version of Parent.

- To call Parent class function inherited

Parentclass.name.functionname(self)

→ Parent-classname.fun-name(self)

S1. Show Name() when we call fun. of Parent

S2. Show Age() class using child class object
then the Rule is :-

- firstly ~~sta~~ called function(2)
is searched in child class
and then in Parent class
if not found in child class
- if called

- if called function is (3)
found in child class
then it will execute
without searching it in
Parent class

- when the object is calling (1)
func. then the func. will
searched in the class of
object type

The child class can call Parent class function By
using ~~super()~~ Super().

Name conflicts in Inheritance Part - 1

1) Instance member variable name conflict

When there is name conflict between instance variable ~~is~~ of base class and derived class then the variable which created first is executed first will created first and only one copy ^{of variable} is their in object. So when second time the variable instance variable encountered we were only accessed ^{the variable} ~~not or~~ ^{or assigned} created instance variable.

2) Static member variable name conflict

When we access static variable (n) ~~as~~ Using derived class (derived.n) then it will always access child static variable in child class. and when we access (n) using Base class (Base.n) then, always access ~~static~~ static variable of Base class. Hiding of variable.

Name conflict in Inheritance Part 2

3) Instance Member function name conflict

Overriding, And hiding.

and hiding. ↗

- Calling function in Parent calls class in overriding)
Super.f1() || Parent class function f1

Overridden function with same name & arg

Hiding function with ~~diff~~ no same name
diff arg.

The child version call Parent version By Using Super().

4) Static member function Name conflict

4) Static member function Name conflict

- Creating object is not necessary to call static method
- Static method can be called using class object.
- When child class object calls static method the child version of it will run.

Name conflict between static and instance member

Static variable and instance variable

- we can access static variable outside the class through object (obj.n) [If there is no instance member]
- If we call through object obj then instance member variable will access and to run static variable we have to call through class object (Derived.n)

Static function and instance function

- There is no concept of fun. overloading in Python
- If the functions of same name is created then the last one will execute.

Accessing Base members in Derived Class [Inheritance]

- 1) Instance member variables of Base class in Derived
- 2) Static member variable of Base class in Derived
- 3) Instance member functions of Base class in Derived
- 4) Static member functions of Base class in Derived

Multiple Inheritance

If a class is derived from more than one class, it is known as Multiple inheritance

Syntax:

Class Base 1:

—
—

Class Base 2:

—
—

Class Derived (Base1, Base2):

—
—

Data Hiding

C++/Java

access specifier/modifier

Private

Protected

Public

Python

class Test:

n = 10 //accessible outside

_n = 200 //hidden var.

- Python changes the name of hidden var. by adding prefix `_classname__n`, (`_Test__n`)
Name Mangling.

Polymorphism

Poly - many
morph - form

one words has different meaning but when used
in sentence it is cleared that which meaning
is to be considered.

e.g.

Class A:

```
def f1(self):  
    print("A - f1")
```

Class B:

```
def f2(self):  
    print("B - f2")
```

obj = A()

obj.f1()

obj = B()

obj.f2()

→ A - f1

→ B - f2

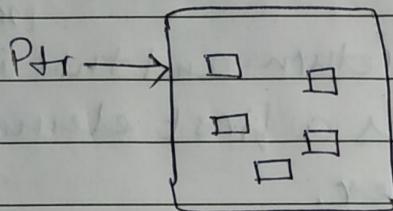
— name — = — main —

Value of name in current program is main
and in imported program is — filename imported

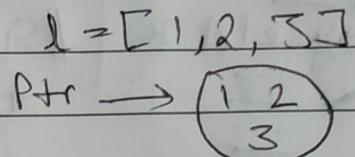
Iterators and Generators

Iterator

- similar like Pointer
- Iterator move from first element to last element and can't move backward
- `it = iter (Sequence)`



Container



for i in l:
 print(i)

→ `li = iter(l)` // list-iterator

`li = iter(l)`

`l = [1, 2, 3]`

`li = iter(l)`

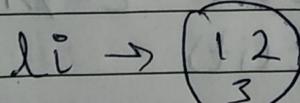
For i in li:

Print(i)

1

2

3



`next(li) → 1`

`next(li) → 2`

`next(li) → 3`

`next(li) → Exception`

while True:

`next(li)`

• through ~~next()~~ we
can access element of
container one by one

• `next()` method return
next value after ~~of~~ value
of Pointed iterator

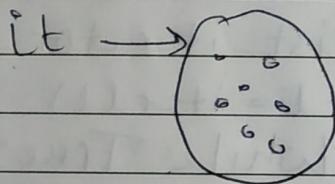
Generators

- Special function
- it gives an iterator
- use of yield keyword it must atleast once

def f1():

~~// Similar as return and this is treated~~
yield x // Similar as return and this is treated
treated as a first element of
container.

it = f1()



def fibonacci(n):
 a, b, c = 0, 1, 0

while True:

if c > n

return

① → Pause →

yield a

a, b = b, a+b

c += 1

it = fibonacci(10)

for i in it:

print(i, end=' ')

it = fibonacci(10)

while True:

print(next(it)).

it = fibonacci(5) ↳

① next(it)

$$\begin{array}{l} a=0 \\ b=1 \\ c=1 \end{array}$$

1 next(it)

1 next(it)

2 next(it)

2 next(it)

3 next(it)

3 next(it)

5 next(it)

Exception next(it)

① next(it) → fun will

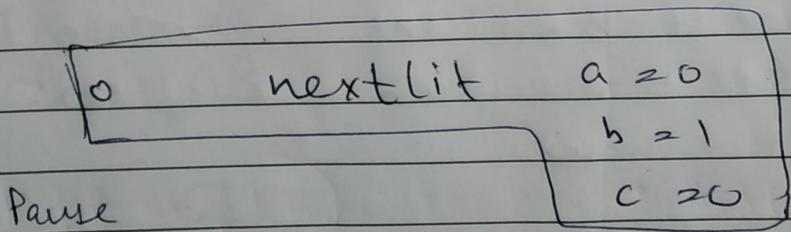
run from
beginning.

`it = fibonaci(5)`

	(1)	(2)
0	<code>next(it)</code>	$a = \varnothing \times \varnothing 2 \mathbb{Z} 8 \mathbb{B}$
1	<code>next(it)</code>	$b = \mathbb{X} \mathbb{X} 2 \mathbb{Z} 8 \mathbb{B} 1 \mathbb{3}$
1	<code>next(it)</code>	$c = \mathbb{D} \mathbb{X} 2 \mathbb{Z} 8 \mathbb{Y} \mathbb{8} 6$
2	<code>next(it)</code>	
3	<code>next(it)</code>	
5	<code>next(it)</code>	
Exception	<code>next(it)</code>	

Note

- Value written after `yield` returns `next()` method.
- the fun. will Pause and values of variable are preserved. and after every `next()` one `next()` method it will resume (the fun.)
- the resumed fun. will resume & stop at `yield`.
- When return run then the Exception is raised "StopIteration"



- The fun. will Pause after "yield a" until `next(it)`

File Handling in Python

File Handling Part - I

The data stored in variable has lifetime of less than life-time time of program or equal, but when program ends the variable destroy. And the data is lost. so to prevent data we have to create a file and save it in secondary storage.

Type of files

1) Text files

- data → sequence of characters
- .txt
- .py
- .dat

2) Binary files

- data → seq. of bytes.
- audio
- video
- image

Logical steps of file handling

- 1) Open file
- 2) Processing
- 3) Close file

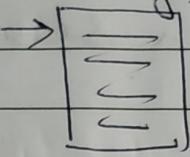
Part - 2

(1) `Open()`

`f = open(filename, mode)`

- 'file.txt' // in current dir.
- 'd:\folder\folder2\file.txt' // absolute path

Modes

- `r` → read only, fileNotFoundError.
- `w` → write only, if no such file then it creates and Erase Previous content.
- `a` → append, No reading, if no such file then it create and can't Erase Previous content
- `x+` → reading, writing, to modify, fileNotFoundError
→  file pointer points at beg. and move to last. and then writes - but we write first then it writes at beg.
- `wt` → reading & writing, if no file then create file, and erase Previous data.
- first write then read, if we first write the data is erased, reading not possible on opening of a file
- `at` → reading and writing, not erased Prev. content and not useful for modify.
- `x` → exclusive write mode
always create new file
use for fresh writing
`X` mode is use when there is no such file of that name.

Modes of binary file.

`rb`, `wb`, `ab`, `tb`, `wtb`, `atb`, `xb`

Part - 3

f = open('file1.txt', 'r')

③ f . close() // closing of a file

② Operations

f = open('file1.txt', 'r')

→ Error : FileNotFoundError

f = open('file1.txt', 'w') // create a file

f . name

→ 'text1.txt'

f . mode

→ 'w'

f . closed

→ False

f . readable()

false

f . writable()

True

Part - 4

- Writing in a file

```
f = open('text1.txt', 'w')  
text = input("Enter some text")  
f.write(text)  
f.close
```

f.writelines(l) // to write sequence

- Reading

read() → read all data
read(n) → read n character
readline() → read one line, in loop read lines
one by one
readlines() → read all lines, and return
list of data.

```
f = open('text.txt', 'r')  
print(f.read()) | s = f.read()  
f.close() | print(s)
```

Using for loop.

```
for line in f:  
    print(line)
```

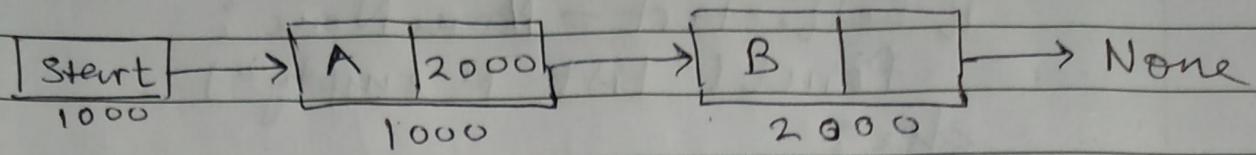
```
f.close()
```

Part 5

Part 6

- How to write object in a file?

linked list



Class node :

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

Class linked list :

```
def __init__(self):  
    self.start = None
```

```
def insertLast(self, value):  
    newNode = Node(value)  
    if self.start == None:  
        self.start = newNode
```

else:

```
    temp = self.start
```

```
    while temp.next != None:
```

```
        temp = temp.next
```

```
    temp.next = newNode
```

```
def deleteFirst(self):
```

```
    if self.start == None:
```

```
        Print("Linked list is empty")
```

else

```
    self.start = self.start.next
```

```
def viewList(self):  
    if self.start == None:  
        print("List is empty")  
    else:  
        temp = self.start  
        while temp.next != None:  
            print(temp.data, end=" ")  
            temp = temp.next
```

```
mylist = LinkedList()  
mylist.insert(10)  
mylist.insert(20)  
mylist.insert(30)  
mylist.insert(40)  
mylist.viewData()  
mylist.deleteFirst()  
mylist.insertData()
```