MODULE 1 : Introduction and Basics
Lecture -3

Today's Target

- **Input()**

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

input()

- Developers often have a need to interact with users, either to get data or to provide some sort of result. Most programs today use a dialog box as a way of asking the user to provide some type of input. While Python provides us with two inbuilt functions to read the input from the keyboard.

- **input (prompt)**
- **raw_input (prompt)**

input()

input (): This function first takes the input from the user and converts it into a string. The type of the returned object always will be <class 'str'>. It does not evaluate the expression it just returns the complete statement as String. For example, Python provides a built-in function called input which takes the input from the user. When the input function is called it stops the program and waits for the user's input. When the user presses enter, the program resumes and returns what the user typed.

EXAMPLE

```
val = input("Enter your value: ")
print(val)
```

Output

```
Enter your name xyz
xyz
```

Input()

```
val = input("Enter your value:\n ")
print(val)
```

Output

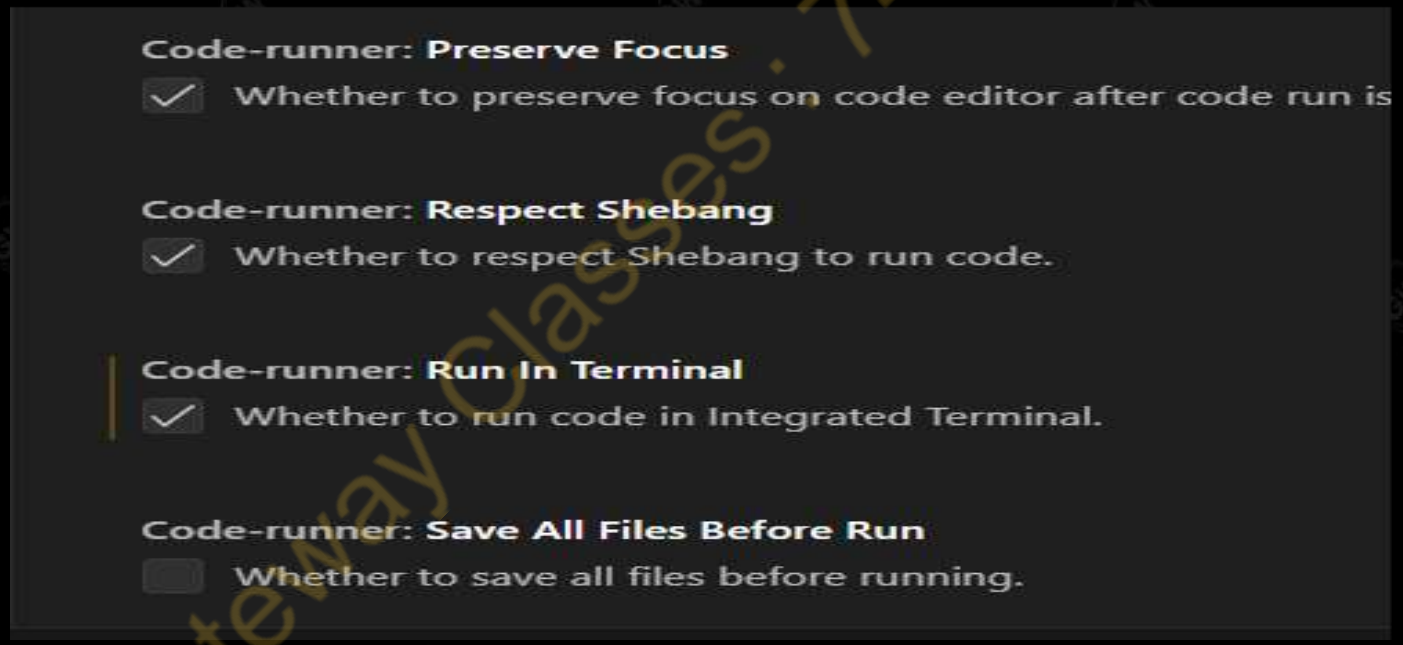
```
Enter your name
xyz
Xyz
```

Another way

Input()

To run the output in terminal

Go to file>preference>setting> write code runner>check the checkbox(run in terminal)



How the input function works in Python :

- When input() function executes program flow will be stopped until the user has given input.
- The text or message displayed on the output screen to ask a user to enter an input value is optional i.e. the prompt, which will be printed on the screen is optional.
- Whatever you enter as input, the input function converts it into a string. if you enter an integer value still input() function converts it into a string. You need to explicitly convert it into an integer in your code using typecasting

Input()

```
number=input("enter the number:")
print(number)
```

Output

```
 enter the number: 123
123
```

Input()

```
number=input("enter the number\n")
print(number)
name=input("enter the name\n")
print(name)
print("type of number", type(number))
print("type of name ", type(name))
```

Input()

output

enter the number

123

123

enter the name

Rani

Rani

type of number <class ‘str’>

Type of name<class ‘str’>

Input()

```
number=int(input("enter the number\n"))
print(number)
name=input("enter the name\n")
print(name)
print("type of number“ , type(number))
print("type of name“ , type(name))
```

Input()

output

enter the number

123

123

enter the name

Rani

Rani

type of number <class ‘int’>

Type of name<class ‘str’>

Input()

```
number=float(input("enter the number in decimal"))
print(number)
print("type of number ", type(number))
```

Output

```
Enter the number in decimal 2.5
2.5
Type of number <class'float'>
```

raw_input()

This function works in older version (like Python 2.x). This function takes exactly what is typed from the keyboard, converts it to string, and then returns it to the variable in which we want to store it.

```
number=raw_input("enter the number in decimal")
print(number)
```

Output

```
raw_input is not defined
```

Write a program to take number from user and add these number

```
x=int(input("enter the first number"))
print(x)
y=int(input("enter the second number"))
z=x+y
print("the sum of two number“ ,z)
```

MODULE 1 : Introduction and Basics
Lecture -4

Today's Target

- Basic Program
- Area and perimeter of rectangle , square
- Area and circumference of circle
- And many more

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

write a program to find out the area of rectangle

```
x=int(input("enter the length of rectangle\n"))
print("the length of rectangle", x)
y=int(input("enter the width of rectangle\n"))
print("the width of rectangle", y)
area=x*y
print("the area of rectangle", area)
```

write a program to find out the area of rectangle

output

enter the length of rectangle

6

the length of rectangle 6

enter the width of rectangle

7

the width of rectangle 7

the area of rectangle 42

write a program to find out the area of rectangle

output

```
Enter the length of rectangle
6.8
Traceback (most recent call last):
  File "/home/main.py", line 1, in <module>
    x=int(input("enter the length of rectangle\n"))
ValueError: invalid literal for int() with base 10: '6.8'
```

write a program to find out the area of rectangle

```
x=float(input("enter the length of rectangle\n"))
print("the length of rectangle", x)
y=float(input("enter the width of rectangle\n"))
print("the width of rectangle", y)
area=x*y
print("the area of rectangle", area)
```

write a program to find out the area of rectangle

output

enter the length of rectangle

6.7

the length of rectangle 6.7

enter the width of rectangle

5.5

the width of rectangle 5.5

the area of rectangle 36.85

write a program to find out the area of rectangle

output

enter the length of rectangle

6

the length of rectangle 6.0

enter the width of rectangle

7

the width of rectangle 7.0

the area of rectangle 42.0

write a program to find out the perimeter of rectangle

```
x=int(input("enter the length of rectangle\n"))
print("the length of rectangle“ , x)
y=int(input("enter the width of rectangle\n"))
print("the width of rectangle", y)
peri=2*(x + y)
print("the perimeter of rectangle“ , peri)
```

write a program to find out the perimeter of rectangle

output

enter the length of rectangle

4

the length of rectangle 4

enter the width of rectangle

5

the width of rectangle 5

the perimeter of rectangle 18

write a program to find out the perimeter of rectangle

```
x=float(input("enter the length of rectangle\n"))
print("the length of rectangle“ , x)
y=float(input("enter the width of rectangle\n"))
print("the width of rectangle", y)
peri=2*(x + y)
print("the perimeter of rectangle“ , peri)
```

write a program to find out the perimeter of rectangle

output

enter the length of rectangle

8.9

the length of rectangle 8.9

enter the width of rectangle

9.8

the width of rectangle 9.8

the perimeter of rectangle 37.400000000000006

write a program to find out the area and perimeter of square

```
a=float(input("enter the side of the square\n"))
print("the side of the square“ , a)
area=a*a
print("the area of the square", area)
perimeter=4*a
print("the perimeter of square“ , perimeter)
```

write a program to find out the area and perimeter of square

output

enter the side of the square

9.8

the side of the square 9.8

the area of the square 96.04000000000002

the perimeter of square 39.2)

write a program to find out the area and circumference of circle

```
r=float(input("enter the radius of circle\n"))
print("the radius of circle“ , r)
area=3.14*r*r
perimeter=2*3.14*r
print("the area of circle", area)
print("the perimeter of circle", perimeter)
```

write a program to find out the area and circumference of circle

output

enter the radius of circle

6.7

the radius of circle 6.7

the area of circle 140.9546

the perimeter of circle 42.076

write a program to swap two number with using third variable

```
x=int(input("enter first number\n"))
y=int(input("enter second  number\n"))
print("the value of x=",x)
print("the value of y=",y)
z=x
x=y
y=z
print("the value of x after swapping", x)print("the value of y after swapping“ ,y)
```

write a program to swap two number with using third variable

enter first number

5

enter second number

7

the value of x= 5

the value of y= 7

the value of x after swapping 7

the value of y after swapping 5

write a program to swap two number without using third variable

```
x=int(input("enter first number\n"))
y=int(input("enter second  number\n"))
print("the value of x=",x)
print("the value of y=",y)
x=x + y
y=x - y
x=x - y
print("the value of x after swapping", x)
print("the value of y after swapping", y)
```

write a program to swap two number without using third variable

enter first number

8

enter second number

9

the value of x= 8

the value of y= 9

the value of x after swapping 9

the value of y after swapping 8

write a program to find out the simple interest

```
p=float(input("enter the principle"))
r=float(input("enter the rate"))
t=float(input("enter the time"))
si=(p*r*t)/100
print("the simple interest", si)
```

write a program to find Celsius to Fahrenheit

```
Celsius = int(input("Enter the Temperature in Celsius :\n"))
 Fahrenheit = (1.8 * Celsius) + 32
 print("Temperature in Fahrenheit :", Fahrenheit)
```

MODULE 1 : Introduction and Basics
Lecture -5

Today's Target

- Data type

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

Data type

- Every value has a data type, and variables can hold values. Python is a powerfully composed language
- A=5 We did not specify the type of the variable a, which has the value five from an integer. The Python interpreter will automatically interpret the variable as an integer.
- We can verify the type of the program-used variable thanks to Python. The type() function in Python returns the type of the passed variable.
- a=5
- print("the type of a ", type(a))

Output

the type of a <class 'int'>

Data type

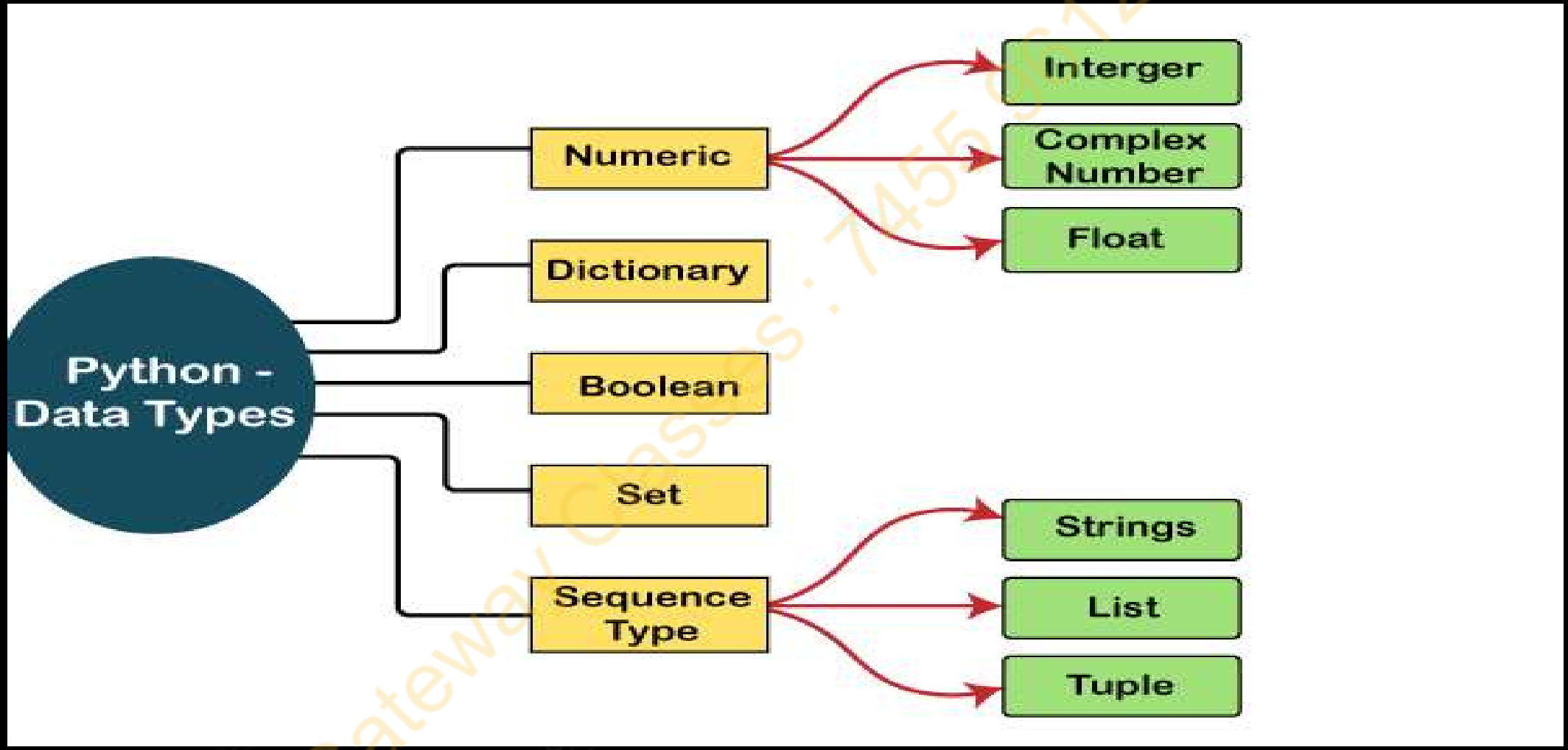
```
a=20.5
b=6
c="hello“
print("thetype of a",type(a))
print("the type of b",type(b))
print("the type of c",type(c))
```

Data type

The following is a list of the Python-defined data types.

- Numbers
- Sequence Type
- Boolean
- Set
- Dictionary

Data type



Numeric Data

- Python, numeric data type is used to hold numeric values.
- Integers, floating-point numbers and complex numbers fall under Python numbers category.
- They are defined as int, float and complex classes in Python.
- int - holds signed integers of non-limited length.
- float - holds floating decimal points and it's accurate up to **15** decimal places.
- complex - holds complex numbers.

Numeric Data

```
num1=5
print(num1,"is type",type(num1))
num2=2.0
print(num2,"is of type",type(num2))
num3=1+2j
print(num3,'is of type',type(num3))
```

Output

```
5 is type <class 'int'>
2.0 is of type <class 'float'>
(1+2j) is of type <class 'complex'>
```

sequencing type

String

The sequence of characters in the quotation marks can be used to describe the string. A string can be defined in Python using single, double, or triple quotes.

```
str1="hi"
str2='bye'
str3='''hello'''
print(str1,"is of type",type(str1))
print(str2,"is of type ",type(str2))
print(str3,"is of type",type(str3))
```

sequencing type

String

output

hi is of type <class 'str'>

bye is of type <class 'str'>

hello is of type <class 'str'>

sequencing type

String handling

```
str1="hello gatewayclasses"
str2="welcome"
print(str1[0:2])#printing first two character using slice operator
print(str2[1:3])#printing second and third character
print(str1[4])#printing the element present at four index
print(str2[5])#printing the element present at fifth index
print(str1*2)#printing the string twice
print(str2*4)#printing the string four time
print(str1+str2)#concatenation
print(str2+str1)
```

sequencing type

String handling

output

he

el

o

m

hello gatewayclasseshello gatewayclasses

welcomewelcomewelcomewelcome

hello gatewayclasseswelcome

welcomehello gatewayclasses

sequencing type

list

List is an ordered collection of similar or different types of items separated by commas and enclosed within brackets [ ]. For example,

```
language=["python","c","java"]
print("the type of language",type(language)
```

Output

```
the type of language <class 'list'>
```

sequencing type

ACCESS ELEMENT IN LIST

To access items from a list, we use the index number **(0, 1, 2 ...)**. For example

```
language=["python","c","java"]
print(language)
print(language[0])
print(language[1])
print(language[2])
```

Output

```
['python', 'c', 'java']
Python
C
java
```

sequencing type

list

```
language=["python","c","java"]

print(language[3])
```

Output

```
ERROR!Traceback (most recent call last):

 File "<string>", line 2, in <module>IndexError: list index out of range
```

sequencing type

list

```
language=["python",1,"c","java",3.5]
print(language)
```

Output

```
['python', 1, 'c', 'java', 3.5]
```

sequencing type

list

```
list1  = [1, "hi", "Python", 2]
 print(type(list1))
 print (list1)
print(list1[2:])#list slicingprint
(list1[3:]) #list slicing print(list1[:2]) #list slicingprint
(list1[0:2])  #list slicing print
 (list1 + list1)  #concatenation using+operatorprint
(list1 * 3)  #list repetition using*operator
```

sequencing type

list

output

[2]

[1, 'hi']

[1, 'hi']

[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]

[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]

Note list data type are mutable

MODULE 1 : Introduction and Basics
Lecture -6

Today's Target

- Data type

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

Tuple data type

- Tuple is an ordered sequence of items same as a list. The only difference is that tuples are immutable.

Tuples once created cannot be modified.

- In Python, we use the parentheses () to store items of a tuple. For example,

sequencing type

Tuple data type

```
product=('microsoft','xbox',499.99)
product2=("samsung","vivo",2,4.555)
print(product)
print(product2)
print("type of product",type(product))
print("type of product",type(product2))
print(product[0])
print(product[1])
print(product2[2])
```

Tuple data type

output

('microsoft', 'xbox', 499.99)

('samsung', 'vivo', 2, 4.555)

type of product <class 'tuple'>

type of product <class 'tuple'>

microsoft

xbox

2

sequencing type

Tuple data type

```
tup  = ("hi", "Python", 2)
# Tuple slicing
print (tup[1:])
print (tup[0:1])
 # Tuple concatenation using + operator
 print (tup + tup)    # Tuple repatation using * operator
 print (tup * 3)
# Adding value to tup. It will throw an error.
tup[2] = "hi"
```

Tuple data type

output

```
('Python', 2)
('hi',)
('hi', 'Python', 2, 'hi', 'Python', 2)
('hi', 'Python', 2, 'hi', 'Python', 2, 'hi', 'Python', 2)
Traceback (most recent call last):
  File "/home/main.py", line 10, in <module>
    tup[2] = "hi"
TypeError: 'tuple' object does not support item assignment
```

Dictionary data type

- dictionary is an ordered collection of items. It stores elements in key/value pairs.
- Here, keys are unique identifiers that are associated with each value

```
# create a dictionary named capital_city
capital_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}
print(capital_city)
```

```
# create a dictionary named capital_city
capital_city = {“Nepal”: “Kathmandu”, “Italy”: “Rome”, “England”: “London”}
print(capital_city)
```

Dictionary data type

- dictionary is an ordered collection of items. It stores elements in key/value pairs.
- Here, keys are unique identifiers that are associated with each value

```
# create a dictionary named capital_city
capital_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}
print(capital_city)
```

```
# create a dictionary named capital_city
capital_city = {“Nepal”: “Kathmandu”, “Italy”: “Rome”, “England”: “London”}
print(capital_city)
```

Dictionary data type

Access Dictionary Values Using Keys

```
# create a dictionary named capital_city
capital_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}

print(capital_city['Nepal'])  # prints Kathmandu

print(capital_city['Kathmandu'])  # throws error message
```

Dictionary data type

```
d={1:'gatewayclasses',2:'python',3:'datatype',4:'module'}
print(d)
print(d.keys())
print(d.values())
print("1name is "+d[1])
print("3name "+d[3])
```

Output

```
dict_keys([1, 2, 3, 4])
dict_values(['gatewayclasses', 'python', 'datatype', 'module'])
1name is gatewayclasses
3name datatype
```

Boolean

The Boolean value can be of two types only i.e. either True or False. The output ***<class 'bool'>*** indicates the variable is a Boolean data type.

```
a = True
type(a)
```

output

```
<class 'bool'>
```

```
b = False
type(b)
```

Output

```
<class 'bool'>
```

sets

- Sets are used to store multiple items in a single variable.
- Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.
- A set is a collection which is *unordered*, *unchangeable**, and *unindexed*.
- *** Note:** Set *items* are unchangeable, but you can remove items and add new items
- thisset = {"apple", "banana", "cherry"}

  print(thisset)
- Output
- {'apple', 'banana', 'cherry'}

sets

- Set items are unordered, unchangeable, and do not allow duplicate values.
- Unordered means that the items in a set do not have a defined order.
- Set items can appear in a different order every time you use them, and cannot be referred to by index or key.
- Set items are unchangeable, meaning that we cannot change the items after the set has been created.
- Once a set is created, you cannot change its items, but you can remove items and add new items.

sets

- DUPLICATES ARE NOT ALLOWED(THEY ARE IGNORED)
- thisset = {"apple", "banana", "cherry", "apple"}
- print(thisset)
- Output
- {'cherry', 'apple', 'banana'}
- thisset = {"apple", "banana", "cherry", True, 1, 2,False,0}
- print(thisset)
- print(type(thisset))
- Output
- {False, True, 2, 'banana', 'cherry', 'apple'}
- <class 'set'>

sets

- set1 = {"apple", "banana", "cherry"}#set item can be of any type
- set2 = {1, 5, 7, 9, 3}
- set3 = {True, False, False}
- print(set1)
- print(set2)
- print(set3)
- set4 = {"abc", 34, True, 40, "male"}
- print(set4)

sets

- output
- {'apple', 'banana', 'cherry'}
- {1, 3, 5, 7, 9}
- {False, True}
- {True, 34, 40, 'male', 'abc'}
- thisset = {"apple", "banana", "cherry",1,1}
- print(len(thisset))
- Output
- 4

MODULE 1 : Introduction and Basics
Lecture -7

Today's Target

- operator

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

operator

- These are the special symbols in Python and are used to execute an Arithmetic or Logical computation. An operator alone cannot perform an activity, it needs an Operand. What is an operand? An Operand is a value that the operator needs to complete a task

Types of Operators in Python

- We have multiple operators in Python, and each operator is subdivided into other operators. Let's list them down and know about each operator in detail.
- Arithmetic operators
- Comparison operators
- Assignment operators

operator

- Logical operators
- Bitwise operators
- Membership operators
- Special operators
  - Identity operators
  - Membership operators

Arithmetic operators

➢Arithmetic operators are used for executing the mathematical functions in Python which include, addition, subtraction, multiplication, division, etc

Operator	Description	Example
Plus	Adds two Operands.	3+3= 6
-Minus	Right-hand Operand is subtracted from the left-hand operand.	20-10=10

Arithmetic operators

Operator	Description	Example
Multiplication	It Multiplies the values on either side of the operands.	10*10 = 100
/ Division	left-hand operand divided by right-hand operand.	50/5 = 10
% modlus	It divides the left-hand operand by the right-hand one and also returns a reminder.	7%2 = 1
// Floor division	Division that results in the whole number being adjusted to the left in the number line.	5.0 / 2 2.5 5.0 //2 2.0
** Exponent	The left operand is raised to the power of the right	10 to the power of 3 10**3=1000

FLOOR DIVISION

➢Floor division is a normal division operation, except it returns the largest possible integer. This integer can either be less than the normal division output or equal to it.

➢The floor function is signified by the ⌊ ⌋ symbol in mathematical terms

Let us now understand the working of the Floor division operation. For example,

⌊36/5⌋

Step 1: Performing the division first. We will divide **36** by **5**.

36 ÷ 5 = 7.2

Step 2: Now, we will perform the floor function on the value we get after division, i.e., **7.2**.

⌊7.2⌋=7

.

FLOOR DIVISION

- Floor division is an operation in Python that allows us to divide two numbers and rounds the resultant value down to the nearest integer. The floor division occurs through the **(//) operator**
- a=543
- b=32
- print(a//b)
- print(a/b)
- output

16.

- 16.97

Arithmetic operators

- x=10
- y=20
- print("x+y=",x+y)
- print("x-y=",x-y)
- print("x*y",x*y)

Output

x+y= 30

x-y= -10

x*y 200

Arithmetic operators

- x=20
- y=10
- print("x/y",x/y)
- x=-20
- y=10
- print("x/y",x/y)
- x=20
- y=-10
- print("x/y",x/y)
- x=-20
- y=-10
- print("x/y",x/y)

Output

x/y 2.0
x/y -2.0
x/y -2.0
x/y 2.0

Arithmetic operators

- x=9
- y=2
- print("x%y=",x%y)
- print("x//y=",x//y))
- x=10
- y=3
- print("x**y=",x**y)

Output

x%y= 1

x//y= 4

x**y= 1000

Python Comparison(Relational) Operators

- The name itself explains that this operator is used to compare different things or values with one another. In Python, the Comparison operator is used to analyze either side of the values and decide the relation between them. The comparison operator is also termed a relational operator because it explains the connection between.

Python Comparison(Relational) Operators

We have different Comparison (Relational) operators. Let’s list them down and get to know each operator in detail.

== Equal

!= Not Equal

> Greater Than

< Less Than

>= Greater Than or Equal to

<= Less Than or Equal

Python Comparison(Relational) Operators

1 Python (==) Equal Comparison Operator

The equal operator is used to compare the two values and to give the result whether they are equal or not.

If both the values are equal, then it gives the result as True, if not False

```
x=10
y=3
print("x==y",x==y)
a=120
b=120
print("a==b",a==b)
```

Output

```
x==y False
a==b True
```

Python Comparison(Relational) Operators

2) Python (!=) Not Equal Comparison Operator

This Not equal operator is used to compare two operands and returns a True statement if they are not equal, and False if they are Equal

```
x=10
y=3
print("x!=y",x!=y)
a=120
b=120
print("a!=b",a!=b)
```

Output

```
x!=y True
a!=b False
```

Python Comparison(Relational) Operators

3)Python (>) greater than Comparison Operator

This operator is mainly used in Python to return true if the Left-hand value is greater than the right one

```
x=10
y=3
print("x>y",x>y)
```

Output

x>y True

Python Comparison(Relational) Operators

4) Python (<) less than Comparison Operator

This Operator compares the right-hand operator with the left-hand one and if the right-hand one is greater than the left-hand one, it returns True statement

```
x=10
y=3
print("x<y",x<y)
```

Output

```
x<y False
```

Python Comparison(Relational) Operators

5) Python(>=) greater than equal to (<=) less than equal to Comparison Operator

This Operator compares the right-hand operator with the left-hand one and if the right-hand one is greater than the left-hand one, it returns True statement

```
x=10
y=3
print("x<=y",x<=y)
print("x>=y",x>=y))
```

Output

```
x<=y  False
x>=y True
```

Python Assignment Operators

The assignment operator is used to assign value to the event, property, or variable. We use this operator in Python to assign values to variables. We have multiple Assignment operators. Let's list them down and know things better.

= Assign Value

+= Add AND

-= Subtract AND

*= Multiply AND

/= Divide AND

%= Modulus AND

**= Exponent AND

//= Floor Division

Python Assignment Operators

```
a=10
print(a)
a+=10
print("a=",a)
a*=10
print("a=",a)
b=5
b/=2
print("b=",b)
b//=2
print("b=",b)
```

Python Assignment Operators

```
b%=2
print("b=",b)
b**=2
print("b=",b)
c=15
c//=2
print("c=",c)
```

Python Assignment Operators

output

a= 20

a= 200

b= 2.5

b= 1.0

b= 1.0

b= 1.0

c= 7

MODULE 1 : Introduction and Basics
Lecture -8

Today's Target

- operator

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

Python logical Operators

Logical operators are used in any programming language to make decisions based on multiple conditions. In Python, we use Logical operators to determine whether a condition is True or False by taking Operand values as a base

And - Logical AND

Or - Logical OR

Not - Logical

Python logical Operators

x	y	x and y	x or y	not(x)	not(y)
0(false)	0	0	0	1	1
0	1	0	1	1	0
1(true)	0	0	1	0	1
1	1	1	1	0	0

Python logical Operators

```
x = 10>2
y = 20>4
print('x and y is',x and y)
print('x or y is',x or y)
print('not(x and y) is',not(x and y))
```

Output

```
x and y is True
x or y is True
not(x and y) is False
```

Python Bitwise operators

- Bitwise operators are used in Python to perform the operations on binary numerals or bit patterns. It operates bit by bit.
- & Binary AND
- | Binary OR
- ^ Binary XOR
- ~ Binary Ones Complement
- << Binary Left Shift
- >> Binary Right Shift

Python logical Operators

x	y	x & y	x |y	X^Y	~(x)	~y)
0(false)	0	0	0	0	1	1
0	1	0	1	1	1	0
1(true)	0	0	1	1	0	1
1	1	1	1	0	0	0

Bitwise AND OPERATOR

	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1
10	0	0	0	0	1	0	1	0
4	0	0	0	0	0	1	0	0
10&4	0	0	0	0	0	0	0	0

Bitwise AND OPERATOR

- a = 10
- b = 4
- print(a & b)
- Output
- 0

Bitwise OR OPERATOR

	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1
10	0	0	0	0	1	0	1	0
4	0	0	0	0	0	1	0	0
10I4	0	0	0	0	1	1	1	0

Bitwise OR OPERATOR

- a = 10
- b = 4
- print(a | b)
- Output
- 14

Bitwise XOR OPERATOR

	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1
10	0	0	0	0	1	0	1	0
6	0	0	0	0	0	1	1	0
10^6	0	0	0	0	1	1	0	0

Bitwise XOR OPERATOR

- a = 10
- b = 6
- print(a ^ b)
- Output
- 12

Bitwise ONES' COMPLEMENT OPERATOR

	2^7 128	**2^6 64**	**2^5 32**	**2^4 16**	**2^3 8**	**2^2 4**	**2^1 2**	**2^0 1**
10	0	0	0	0	1	0	1	0
~10	1	1	1	1	0	1	0	1

Bitwise XOR OPERATOR

- a = 10
- print(~a)
- Output
- -11

Bitwise left shift OPERATOR

	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1
10	0	0	0	0	1	0	1	0
10<<1	0	0	0	1	0	1	0	0(extra bit added)

Bitwise XOR OPERATOR

- a = 10

print(a <<1)

Print(a<<2)

- Output
- 20
- 40

Bitwise right shift OPERATOR

	2^7 128	2^6 64	2^5 32	2^4 16	2^3 8	2^2 4	2^1 2	2^0 1
10	0	0	0	0	1	0	1	0
10>>1	0(extra bit added)	0	0	0	0	1	0	1

Bitwise XOR OPERATOR

- a = 10

print(a >>1)

Print(a>>2)

- Output
- 5
- 2

Python membership Operators

- Python Membership operators
- Membership Operators are mainly used in Python to find whether a value or variable is present in a sequence (string, tuple, list, set, and directory). “In” and “Not In” are the two membership operators available in Python.

Python membership Operators

➢Python In Membership Operator

➢This In operator is used in Python to evaluate if a particular value is there or not in a sequence. The In operator returns True if it is the specified element found in the list, otherwise, it returns false.

```
list=[1,2,3,4,5]
b=2
if b in list:
print("present")
else:
print("not")
```

Output

```
present
```

Python membership Operators

2 python Not in Membership Operator

“Not in” operator, the name itself expresses the meaning that something is not inside. It goes through a particular sequence to find whether a value is in the specified list or not. If it finds there is no such value, it returns True, otherwise False

```
list=[1,2,3,4,5]
b=2
if b not in list:
print("hi")
else:
 print("bye")
```

Output bye

Identity Operators

Identity operators are used to compare the objects if both the objects are actually of the same data type and share the same memory location.

There are different identity operators such as

'is' operator – Evaluates to True if the variables on either side of the operator point to the same object and false otherwise.

```
x=5
y=5
print(x is y)
```

Output

```
true
```

Identity Operators

```
string1 = "Hello"

string2 = "Hello“

Z=string1

print(string1 is string2)

print(id(string1))

print(id(string2))

Print(id)
```

Output

```
True

140025108758000

140025108758000

140025108758000
```

Identity Operators

```
list1 =[10,20,30]
list2 =[10,20,30]
print(list1 is list2)
print(id(list1))
print(id(list2))#is operator work only on integer and string
```

Output

False

140013677373632

140013677480128

Identity Operators

```
x = ["Gb", "for", "Gb"]

y = ["Gb", "for", "Gb"]

z = x

print(x is not z)

# Returns True because x is not the same object as y,

# even if they have the same content

print(x is not y)

# To demonstrate the difference between "is not" and "!=":

# This comparison returns False because x is equal to y

print(x != y)
```

MODULE 1 : Introduction and Basics
LEC 9

Today's Target

- **OPERATOR PRECEDENCE AND ASSOCIATIVITY**

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

AKTU QUESTION

Q1	Write a short note on operator precedence Or Explain operator precedence and associativity in detail	AKTU2019-20 AKTU 2021-22
Q2	Give the solution of the following ➢ a & b << 2 //5 * *2+c^b ➢ a**2<<2>>b**2^c**3 ➢ a=3,b=5,c=10	AKTU 2022-23

Operator precedence and associativity

- In Python, operators have different levels of precedence, which determine the order in which they are evaluated. When multiple operators are present in an expression, the ones with higher precedence are evaluated first. In the case of operators with the same precedence, their associativity comes into play, determining the order of evaluation

Operator precedence and associativity

Python Operator Precedence

Precedence	Operator Sign	Operator Name
Highest	**	Exponentiation
	+x, -x, ~x	Unary positive, unary negative, bitwise negation
	*, /, //, %	Multiplication, division, floor, division, modulus
	+, -	Addition, subtraction
	<<, >>	Left-shift, right-shift
	&	Bitwise AND
	^	Bitwise XOR
	|	Bitwise OR
	==, !=, <, <=, >, >=, is, is not	Comparison, identity
	not	Boolean NOT
	and	Boolean AND
Lowest	or	Boolean OR

Operator precedence and associativity

Operators	Description	Associativity
()	Parentheses	Left to right
x[index], x[index : index]	Subscription, slicing	Left to right
await x	Await expression	N/A
**	Exponentiation	Right to left
+x, -x, ~x	Positive, negative, bitwise NOT	Right to left
*, @, /, //, %	Multiplication, matrix, division, floor division, remainder	Left to right
+, –	Addition and subtraction	Left to right

Operator precedence and associativity

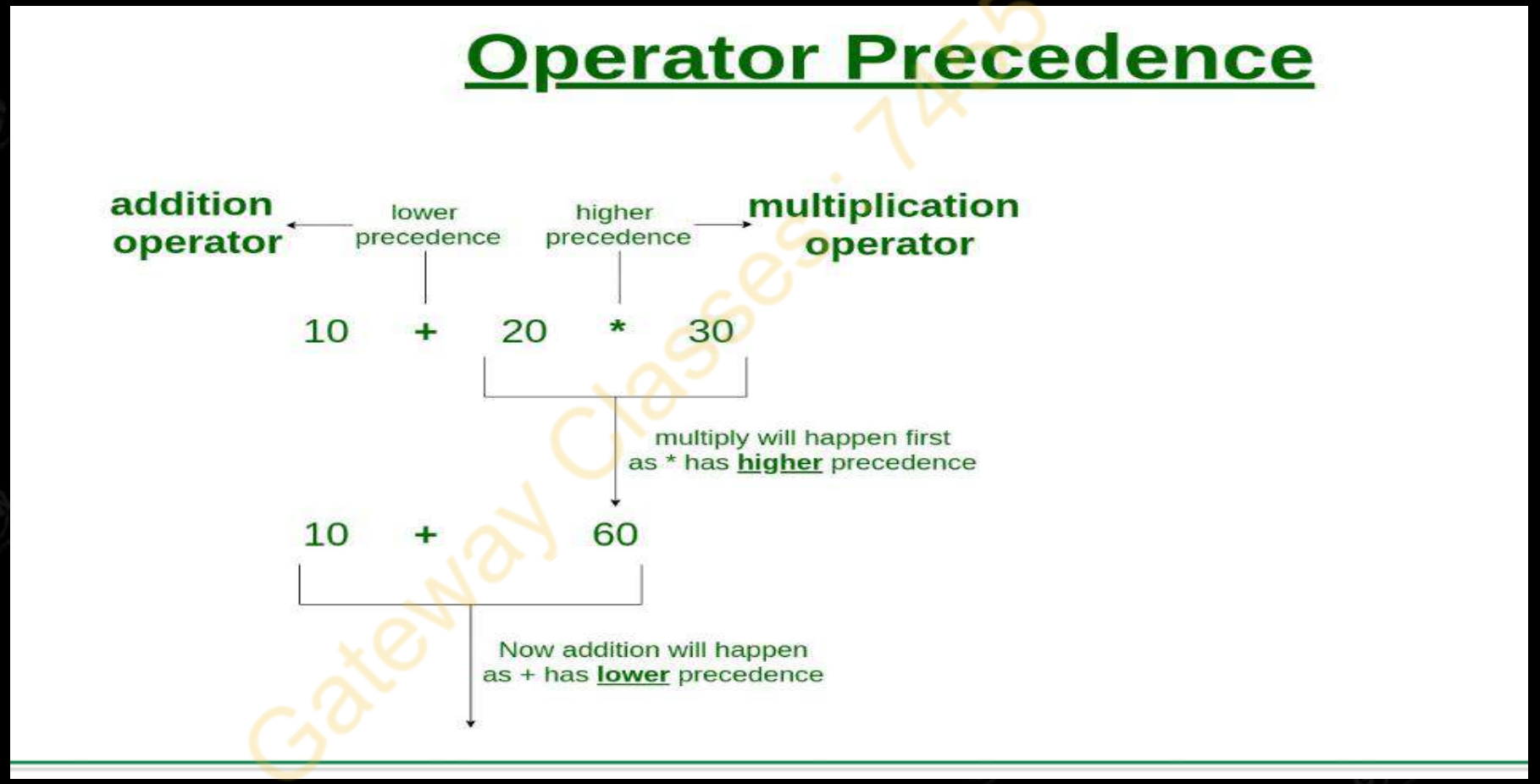
Operators	Description	Associativity
<<, >>	Shifts	Left to right
&	Bitwise AND	Left to right
^	Bitwise XOR	Left to right
|	Bitwise OR	Left to right
in, **not in**, **is**, **is not**, <, <=, >, >=, !=, ==	Comparisons, membership tests, identity tests	Left to Right
not x	Boolean NOT	Right to left
and	Boolean AND	Left to right

Operator precedence and associativity

Operators	Description	Associativity
or	Boolean OR	Left to right
if-else	Conditional expression	Right to left
lambda	Lambda expression	N/A
:=	Assignment expression (walrus operator)	Right to left

Precedence of Python Operators

- This is used in an expression with more than one operator with different precedence to determine which operation to perform first..



Precedence of Python Operators

- # Precedence of '+' & '*'
- expr = 10 + 20 * 30
- print(expr)
- OUTPUT
- 610
- X=8>6>6
- print(X)
- OUTPUT
- FALSE

Precedence of Logical Operators in Python

```
name="Alex"
age=0
if name=="Alex" or name=="john" and age>=2:
    print("hello")
else:
    print("bye")
```

Output

```
hello
```

Precedence of Logical Operators in Python

```
name="Alex"
age=0
if (name=="Alex" or name=="john“) and age>=2:
    print("hello")
else:
    print("bye")
```

Output

bye

Associativity of Python Operators

- If an expression contains two or more operators with the same precedence then Operator Associativity is used to determine. It can either be **L**eft to **R**ight or from **R**ight to **L**eft.

- 100/10*10
- 10*10
- 100

Associativity of Python Operators

- print(100 / 10 * 10)
- #left associative
- print(5 - 2 + 3)#left associative
- print(5 - (2 + 3))#()is higher precedence -,+
- print(2 ** 3 ** 2)# ** is right associativity
- Output
- 100.0
- 6
- 0
- 512

Operators Precedence and Associativity in Python

- Operators Precedence and Associativity are two main characteristics of operators that determine the evaluation order of sub-expressions in the absence of brackets.
- 100+200/10-3*10
- 100+20-3*10
- 100+20-30
- 120-30
- 90

```
expression = 100 + 200 / 10 - 3 * 10
print(expression)
```

Output 90.0

Non Associative Operators

- In Python, most operators have associativity, which means they are evaluated from left to right or right to left when they have the same precedence. However, there are a few operators that are non-associative, meaning they cannot be chained together.
- a = 5
- b = 10
- c = 15
- a = b = (a < b) += (b < c)
- Output
- throw an error

Non Associative Operators

- # Defining the variables a, b, and, c
- a = b = c = 3

- #If the operator += follows associativity answer will shown otherwise error will be raised
- a = b = c += 2

note

- X=7 formula number*2^n where n=no of bits you want to shift
- X<<1
- 7*2^1=14
- X<<3
- 7*2^3=56
- Y=7
- Y>>1 formula number/2^n n=no of bits you want to shift
- 7/2^1
- 3
- Y>>2 = 7/2^2=1

AKTU QUESTIION

- Solve the following question step by step
- a & b << 2 //5 ** 2+c^b
- a=3,b=5,c=10
- 3 &5 << 2 // 5 **2 +10 ^5 ((1) ** (2) // (3) + (4) << (5) & (6) ^ precedence)
- 3 & 5 << 2 // 25 +10^5
- 3 & 5 << 0 +10 ^ 5
- 3 & 5<<10 ^5 5*2^10=5*1024=5120
- 3&5120^5
- 0^5 =5 (note a^0=a a^1= a')

AKTU QUESTION

- Solve the following question step by step
- a**2<<2>>b**2^c**3
- a=3,b=5,c=10
- precedence 1)**(right to left) 2) <<,>>(left to right) 3) ^(left to right)-
- 3 ** 2 << 2 >> 5 ** 2 ^ 10 ** 3
- 3 ** 2 << 2 >> 5 ** 2 ^1000
- 3 **2 << 2 >> 25 ^ 1000
- 9 << 2 >>25 ^1000
- 36>>25^100
- 0^1000
- 1000

MODULE 1 : Introduction and Basics
Lecture -10

Today's Target

- Type conversion

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

AKTU QUESTION

Q1	What is type conversion? Explain its types Or Write a short note on type conversion	AKTU 2019-20 AKTU 2017-18

Type conversion in python

- The act of changing an object's data type is known as type conversion. The Python interpreter automatically performs Implicit Type Conversion.
- Python prevents Implicit Type Conversion from losing data. The user converts the data types of objects using specified functions in explicit type conversion, sometimes referred to as type casting. When type casting, data loss could happen if the object is forced to conform to a particular data type
- Two type of type conversion is there –
- Implicit type
- Explicit type(type casting)

Implicit Type Conversion in Python

- In Implicit type conversion of data types in Python, the Python interpreter automatically converts one data type to another without any user involvement

```
x =10
print("x is type of",type(x))
y =10.6
print("y is type of",type(y))
z=x + y
print(z)
print("z is of type",type(z))
```

Output

```
x is type of <class 'int'>
y is type of <class 'float'>
20.6
z is of type <class 'float'>
```

Explicit Type Conversion in Python

➢In Explicit Type Conversion in Python, the data type is manually changed by the user as per their requirement. With explicit type conversion, there is a risk of data loss since we are forcing an expression to be changed in some specific data type.

Explicit Type Conversion in Python

Converting integer to float

- **int(a, base)**: This function converts **any data type to an integer**. 'Base' specifies the **base in which the string is** if the data type is a string.

float(): This function is used to convert **any data type to a** floating-point **number.**

```
s="10010“
c=int(s,2)
print("after converting to integer base :",c)
e=float(c)
print("after converting to float",e)
f=float(s)
print("after converting to float from string",f)
```

output

```
after converting to integer base : 18
after converting to float 18.0
after converting to float from string 10010.0
```

Explicit Type Conversion in Python

Python Type conversion using ord(), hex(), oct()

ord(): This function is used to convert a character to an integer.

hex(): This function is to convert an integer to a hexadecimal string.

oct(): This function is to convert an integer to an octal string

NOTE

Python does not have a character data type, a single character is simply a string with a length of 1.

Explicit Type Conversion in Python

```
s="4“
c=ord(s)
print("after converting string to integer",c)#Ascii value of 4 IS 52
c=hex(56)
print("after converting integer to hexadecimal",c)
c=oct(56)
print("after converting integer to octadecimal",c)
```

Output

```
after converting string to integer 52
after converting integer to hexadecimal 0x38
after converting integer to octadecimal 0o70
```

python type conversion using tuple(), set(), list()

```
s = 'python'
c=tuple(s)
print("after converting string to tuple",c)
print(type(c))
c= set(s)
print("after converting string to set",c)
print(type(c))
c=list(s)
print("after converting string to set",c)
print(type(c))
```

python type conversion using tuple(), set(), list()

OUTPUT

```
after converting string to tuple ('p', 'y', 't', 'h', 'o', 'n')
<class 'tuple'>
after converting string to set {'h', 'p', 'y', 'n', 't', 'o'}
<class 'set'>
after converting string to set ['p', 'y', 't', 'h', 'o', 'n']
<class 'list'>
```

Convert ASCII value to characters

```
a = chr(76)
b = chr(77)

print(a)
print(b)
```

OUTPUT

```
L
M
```

Python code to demonstrate Type conversion using dict(), complex(), str()

dict(): This function is used to convert a tuple of order (key, value) into a dictionary.

str(): Used to convert an integer into a string.

complex(real,imag) : This function converts real numbers to complex(real,imag) number.

Python3

Python code to demonstrate Type conversion using dict(), complex(), str()

```
a = 1
b = 2
 tup = (('a', 1) ,('f', 2), ('g', 3))
 c = complex(1,2)
print ("After converting integer to complex number : ",c)
 c = str(a)
print ("After converting integer to string : ",c)
 c = dict(tup)
print ("After converting tuple to dictionary :",c)
```

Python code to demonstrate Type conversion using dict(), complex(), str()

Output

After converting integer to complex number : (1+2j)

After converting integer to string : 1

After converting tuple to dictionary : {'a': 1, 'f': 2, 'g': 3}

MODULE 1 : Introduction and Basics
Lecture -11

Today's Target

- Program cycle for python
- IDE

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

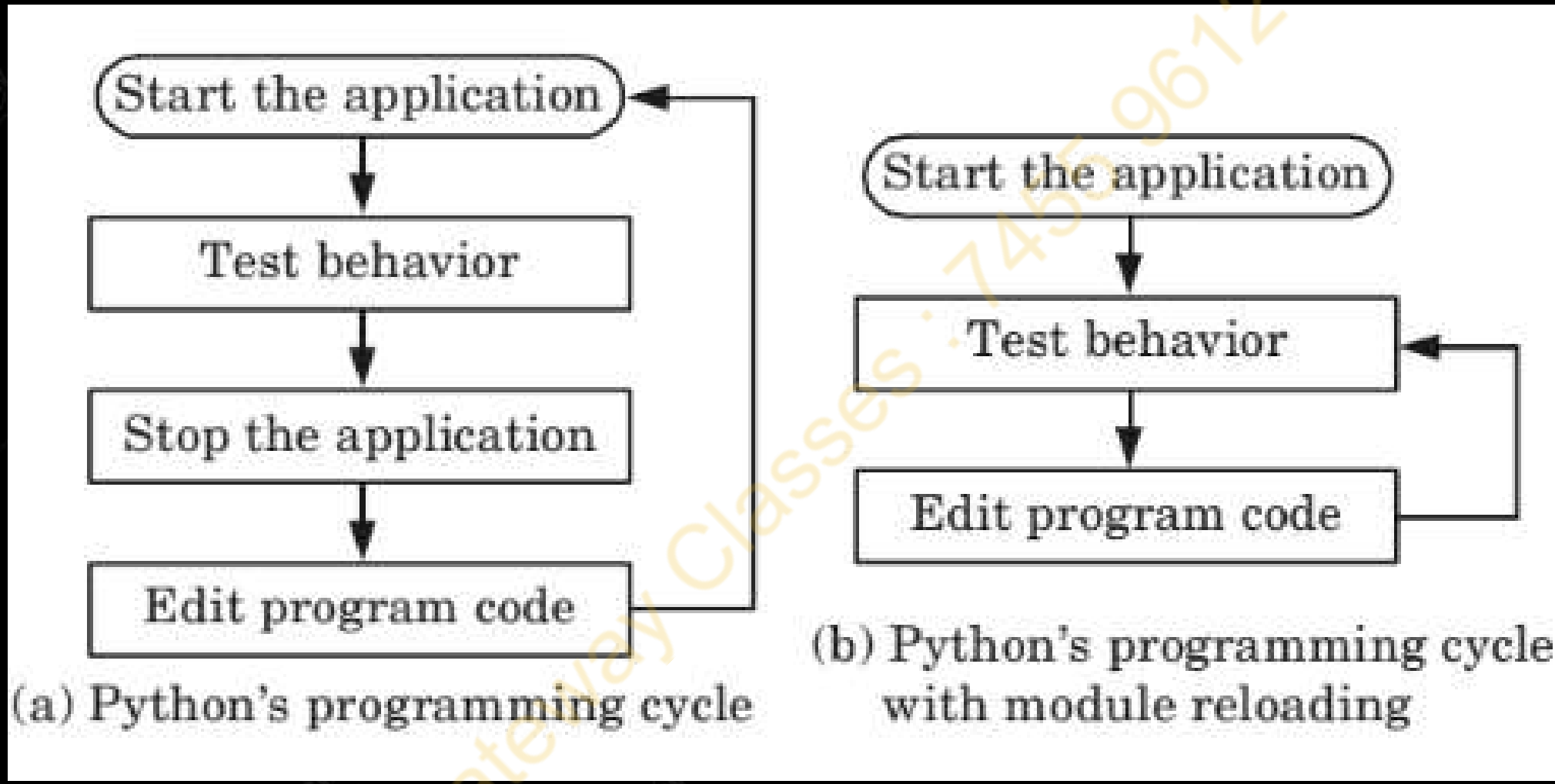
AKTU QUESTION

Q1	**Write a short note programming cycle for python**	**AKTU 2019-20** **AKTU 2021-22**
Q2	Explain features of two python IDES’	**AKTU 2022-23**

Programming cycle for python

- Python's programming cycle is dramatically shorter than that of traditional programming cycle.
- In Python, there are no compile or link steps.
- Python programs simply import modules at runtime and use the objects they contain. Because of this, Python programs run immediately after changes are made.
- In cases where dynamic module reloading can be used, it is even possible to change and reload parts of a running program without stopping it at all.
- Python's impact on the programming cycle is as follows:

Programming cycle for python



(a) Python's programming cycle

(b) Python's programming cycle with module reloading

Programming cycle for python

➢Since Python is interpreted, there is a rapid turnaround after program changes. And because Python's parser is embedded in Python-based systems, it is easy to modify programs at runtime.

What is IDE

➢An integrated development environment (IDE) is a software application that helps programmers develop software code efficiently. It increases developer productivity by combining capabilities such as software editing, building, testing, and packaging in an easy-to-use application. Just as writers use text editors and accountants use spreadsheets, software developers use IDEs to make their job easier.

Why IDE is important ?

➢**Code editing automation**

Programming languages have rules for how statements must be structured. Because an IDE knows these rules, it contains many intelligent features for automatically writing or editing the source code.

➢**Syntax highlighting**

An IDE can format the written text by automatically making some words bold or italic, or by using different font colors. These visual cues make the source code more readable and give instant feedback about accidental syntax errors.

➢**Intelligent code completion**

Various search terms show up when you start typing words in a search engine. Similarly, an IDE can make suggestions to complete a code statement when the developer begins typing.

Why IDE is important ?

➢ **Refactoring support**

Code refactoring is the process of restructuring the source code to make it more efficient and readable without changing its core functionality. IDEs can auto-refactor to some extent, allowing developers to improve their code quickly and easily. Other team members understand readable code faster, which supports collaboration within the team.

➢ **Local build automation**

IDEs increase programmer productivity by performing repeatable development tasks that are typically part of every code change. The following are some examples of regular coding tasks that an IDE carries out.

.

.

Why IDE is important ?

➢ **Testing**

The IDE allows developers to automate unit tests locally before the software is integrated with other developers' code and more complex integration tests are run.

➢ **Debugging**

Debugging is the process of fixing any errors or bugs that testing reveals. One of the biggest values of an IDE for debugging purposes is that you can step through the code, line by line, as it runs and inspect code behavior. IDEs also integrate several debugging tools that highlight bugs caused by human error in real time, even as the developer is typing.

Some python IDE

- PyCharm
- Spyder
- Pydev
- IDLE
- VISUAL STUDIO

p

pyCharm

GATEWAY CLASSES

➢PyCharm is a hybrid platform developed by JetBrains as an IDE for Python. It is commonly used for Python application development. Some of the unicorn organizations such as Twitter, Facebook, Amazon, and Pinterest use PyCharm their Python IDE

➢Features

1. Intelligent Code Editor:

➢It helps us write high-quality codes!

➢It consists of color schemes for keywords, classes, and functions. This helps increase the readability and understanding of the code.

➢It helps identify errors easily.

➢It provides the auto complete feature and instructions for the completion of the code

p

pyCharm

GATEWAY CLASSES

Code Navigation:

- It helps developers in editing and enhancing the code with less effort and time.
- With code navigation, a developer can easily navigate to a function, class, or file.
- A programmer can locate an element, a symbol, or a variable in the source code within no time.
- Using the lens mode, further, a developer can thoroughly inspect and debug the entire source code.

Refactoring

- It has the advantage of making efficient and quick changes to both local and global variables.
- Refactoring in PyCharm enables developers to improve the internal structure without changing the external performance of the code.

.

Assistance for Many Other Web Technologies:

- It helps developers create web applications in Python.
- It supports popular web technologies such as HTML, CSS, and JavaScript.
- Developers have the choice of live editing with this IDE. At the same time, they can preview the created/updated web page. The developers can follow the changes directly on a web browser.
- PyCharm also supports AnglularJS and NodeJS for developing web applications.

5. Support for Popular Python Web Frameworks

- PyCharm supports web frameworks such as Django.
- It provides the autocomplete feature and suggestions for the parameters of Django.
- It helps in debugging the codes of Django.
- It also assists web2py and Pyramid, the other popular web frameworks.

SPYDER

GATEWAY CLASSES

Spyder :

Spyder is widely used for data science works. It is mostly used to create a secure and scientific environment for Python. Spyder Python uses PyQt (Python plug-in) which a developer can add as an extension.

•Features of Spyder :

- It has good syntax highlighting and auto code completion
- Spyder Python explores and edits variables directly from GUI.
- It performs very well in multi-language editor.

p

SPYDER

GATEWAY CLASSES

Features of Spyder :

- Availability of breakpoints (debugging and conditional breakpoints)
- Automatic colon insertion after if, while, etc

PYDEV

GATEWAY CLASSES

PyDev :

- **P**yDev is an open-source integrated development environment (IDE) for Python.
- It is designed to provide a complete development environment for Python programmers.
- Also, it is built on top of the Eclipse platform and supports various features like debugging, code analysis, code completion, and much more.
- PyDev is compatible with all the major operating systems like Windows, Linux, and Mac OS X, making it a popular choice among developers..

PYDEV

GATEWAY CLASSES

- **Code completion:** PyDev provides intelligent code completion. With the help of this, we can write code faster and with fewer errors. It suggests the available methods and variables for a particular object or module.
- **Debugging:** PyDev has a built-in debugger. It allows us to step through our code and identify and fix errors quickly.
- **Code analysis:** PyDev performs static code analysis. It is used to identify potential errors and provide suggestions for improvement, which can help us write better code.

p

PYDEV

GATEWAY CLASSES

➢ **Multi-platform support:** PyDev is compatible with all major operating systems, including Windows, Linux, and Mac OS X, making it a popular choice among developers.

➢ **Integration with Eclipse:** PyDev is built on top of the Eclipse platform, which provides a rich set of features for development, including support for other programming languages and version control systems.

➢ **Plugins:** PyDev supports plugins, which can extend its functionality to support additional features, such as Django and Flask web frameworks.

IDLE

IDLE :

- IDLE is a basic IDE mainly used by beginner level developer.
- IDLE Python is a cross-platform IDE, hence it increases the flexibility for users.
- It is developed only in Python in collaboration with Tkinter GUI toolkit.
- The feature of multi-window text editor in IDLE has some great functions like smart indentation, Python colorizing, and undo option

p

IDLE

GATEWAY CLASSES

IDLE :

- Python IDLE's code editor has features like syntax highlighting and code completion that make it simpler and faster to write Python programmers.
- Python IDLE has a built-in debugger that enables programmers to walk through their code and find faults and problems.
- Python IDLE may be used on Linux, macOS, and Windows thanks to its cross-platform nature.
- Python IDLE is included with the Python installation, thus users don't need to install any more programmes in order to begin coding in Python

VISUAL STUDIO

➢**Visual studio :** It enables development for various platforms and has its own marketplace for extensions.

Features of visual studio :

➢i. It supports Python coding in visual studio, debugging, and other activities.

➢ii. It has both paid and free versions in the market with great features.

➢A visual studio code is a lightweight software application with a powerful source code editor that runs on the desktop. It is a free source code editor developed by Microsoft for Windows, Mac OS and Linux. It is a software editor that has a rich extension of various languages like C++, C+, C, Java, Python, PHP, Go, etc. and runtime language extensions such as .NET and Unity. It is easy to edit, build, syntax highlighting, snippets, code refactoring and debugging.

MODULE 1 : Introduction and Basics
Lecture -12

Today's Target

- python is considered as interpreted language
- Element of python
- What happen when every statement with semicolon

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

AKTU QUESTION

Q1	Explain why python is considered as interpreted language or How python an interpreted language	AKTU 2021-22 AKTU 2020-21
Q2	What type of language is python	AKTU 2019-20
Q3	In some language every statement end with a semicolon(;) what happen when you put a semicolon at the end of python statement	AKTU 2019-20
Q4	what are the Element of python	AKTU 2019-20
Q5	What is python? How python is interpreted? What are the tools to find bugs or perform static analysis	AKTU 2019-20 AKTU 2017-18

python is considered as interpreted language

What is an Interpreted Language?

- Any programming language that isn’t already in "machine code" before runtime is considered an interpreted language. An interpreted language is a computer programming language in which the instructions are executed without first being compiled into machine instructions.
- In other words, unlike compiled languages, an interpreted language does not require prior translation. The translation occurs concurrently with the program’s execution.
- Instead of being instantly executed by the target computer, the instructions are read and executed by another program. Some Interpretable scripting languages include JavaScript, Perl, Python, BASIC, and others.

python is considered as interpreted language

Why Python is Interpreted Language?

- The most common saying, which is also published in various books, is that Python is an interpreted language, however, the hidden fact is that Python is both compiled and interpreted. This might seem contradictory at first, but it's important to understand how Python's compilation and interpretation work together.
- When you write a Python program, it is initially in source code form. This source code needs to be transformed into an executable form that can be executed by the computer. There are two ways to do this:

python is considered as interpreted language

compilation: In this process, the source code is translated into machine code or bytecode ahead of time. The resulting compiled code can be executed directly without any further processing. Python uses a compiler to transform the source code into bytecode, which can be executed by the Python virtual machine. This is how Python code is executed in environments like PyCharm, Visual Studio Code, or other integrated development environments (IDEs) that use Python's static analysis tools.

Interpretation: In this process, the source code is interpreted and executed directly by the interpreter at runtime. When you run a Python script from the command line or an interactive shell, the interpreter reads the source code and executes it line by line. This is how Python code is executed in interactive environments like Jupyter notebooks, or in web applications where Python code is executed on the server side.

python is considered as interpreted language

- Python’s compilation and interpretation work together to provide a flexible and dynamic programming environment. When Python code is compiled, the resulting bytecode can be cached and reused for faster execution in the future. This also allows for some level of static analysis and optimization of the code. When Python code is interpreted, it allows for rapid development and debugging, and provides a more interactive programming experience

Advantages of Interpreted Language

- **Portability:** Interpreted languages are typically platform-independent, which means that they can run on different operating systems without modification. This is because the interpreter itself is usually written in a low-level language that is compiled for each platform.
- **Easy to learn and use:** Interpreted languages typically have simple syntax and a smaller set of language constructs, making them easier to learn and use than compiled languages
- **Rapid development:** Because there is no need to compile or link the code before running it, interpreted languages offer a faster development cycle than compiled languages. Changes to the code can be tested and seen immediately, which is especially useful in a development environment.

Advantages of Interpreted Language

Flexibility: Interpreted languages are often more flexible than compiled languages, as they allow for dynamic typing and more dynamic programming constructs such as reflection and introspection. This can make it easier to write code that adapts to changing requirements or situations.

Debugging: Debugging can be easier in an interpreted language because error messages often include detailed information about the line of code where the error occurred. Additionally, many interpreted languages include debugging tools that allow developers to step through code and inspect variables at runtime.

Interoperability: Interpreted languages can often interoperate with other languages more easily than compiled languages. For example, Python can call C code using the ctypes module, and JavaScript can be embedded in HTML pages

Disadvantages of Interpreted Language

- **Slower execution speed:** Interpreted languages are generally slower than compiled languages because the code is translated into machine instructions on-the-fly at runtime, rather than being compiled ahead of time into optimized machine code. This can make them less suitable for applications that require high performance, such as video games or scientific simulations.
- **Lack of static type checking:** Interpreted languages often use dynamic typing, which means that variable types are determined at runtime rather than being declared in advance. This can make it harder to catch errors related to type mismatches or incorrect function calls until runtime.
- **Security concerns:** Interpreted languages can be more vulnerable to security issues such as cross-site scripting (XSS) or SQL injection attacks. This is because the code is executed in the context of the interpreter, which can make it easier for attackers to exploit vulnerabilities in the interpreter itself.

python as a Interpreted Language

Q1 – How does the Python interpreter work?

A – The Python interpreter reads the source code line by line and then executes each line of code one at a time. It translates the source code into bytecode, which is then executed by the Python virtual machine.

Q2 – Can Python be compiled?

A – Yes, Python can be compiled into bytecode or into native machine code.

Q3 – Is Python slower than compiled languages like C or C++?

A – Yes, Python is generally slower than compiled languages like C or C++. This is because the interpreter has to translate the source code into bytecode at runtime, which takes more time than compiling the code ahead of time into machine code.

What happen when you put semicolon at the end of python statement

- The common meaning of a semicolon(;) in various programming languages is to end or discontinue the current statement. In programming languages such as C, C++, and Java, it is necessary to use a semicolon to end a line of code. However, this is not the case with Python.
- A semicolon in Python signifies separation rather than termination. It allows you to write multiple statements on a single line

Print(“hi”);print(“hello”);print(“bye”)

Output

hi

hello

bye

What happen when you put semicolon at the end of python statement

- In this example, we'll try to loop through two statements separated by a semicolon and see if the loop prints both or just the first one
- for I in range (4):
- print('hi');
- Print("hello");

OUTPUT
Hi
Hello
Hi
Hello
Hi
Hello
Hi
Hello

What happen when you put semicolon at the end of python statement

- **Python will throw an error if you use a semicolon to separate a normal expression from a block statement i.e loop.**
- print('Hi') ; for i in range (4): print ('Hello')
- Output
- Error
- Conclusion

  semicolon in Python is mostly used to separate multiple statements written on a single line.
- A semicolon is used to write a minor statement and reserve a bit of space like name = Marie; age = 23; print(name, age)
- The use of semicolons is very “non-pythonic” and it is recommended not to be used unless you desperately need it.

What type of language is python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics developed by Guido van Rossum. It was originally released in 1991. Designed to be easy as well as fun, the name "Python" is a nod to the British comedy group Monty Python.

Element of python

These are the basic element of python

Datatype

Operator

Variable

Function

keywords

Tools to find bugs and perform static analysis

- Writing efficient, reliable, and secure code is highly valued in software development.
- With the popularity of Python programming language skyrocketing in recent years, it has become increasingly important for developers to ensure their code is error-free, secure, and optimized.
- However, manually inspecting every line of Python code can be daunting and time-consuming. That is where static code analysis comes into play.
- Static code analysis is the procedure of inspecting application source code without executing it.
- It detects potential errors, security flaws, dependencies, bugs, and other issues in the codebase.

Benefits of Using Static Code Analysis for Python

- **Identify bugs in early stages:**

It can help identify bugs, type checks, and common security issues early in the software development cycle, reducing the risk of expensive and time-consuming fixes later.

- **Enforces Coding Standards:**

By enforcing coding standards, style guides, and best practices, static code analysis can help produce more maintainable code that is easier to understand and debug.

Benefits of Using Static Code Analysis for Python

- **Returns Instant Feedback:**

You can get instant feedback on code improvements and alerts on external dependencies. In addition, this process can highlight areas of the code that may be difficult to read or understand, leading to better documentation and more readable code.

- **Reduce Technical Debt:**

Using static code analysis reduces technical debt by specifying code areas that are inefficient, challenging to maintain, or pose security risks.

Static code analysis tools for python

- **Pychecker** and **Pylint** are the static analysis tools that help to find bugs in python.
- **Pychecker** is an opensource tool for static analysis that detects the bugs from source code and warns about the style and complexity of the bug.
- **Pylint** is highly configurable and it acts like special programs to control warnings and errors, it is an extensive configuration file Pylint is also an opensource tool for static code analysis it looks for programming errors and is used for coding standard. it checks the length of each programming line. it checks the variable names according to the project style. it can also be used as a standalone program, it also integrates with python IDEs such as Pycharm, Spyder, Eclipse, and Jupyter
- **Pychecker** can be simply installed by using pip package pip install Pychecker if suppose if you use python 3.6 version use upgrade pip install Pychecker --upgrade Pylint can be simply installed by using pip package
- **pip install Pylint**

MODULE 1 : Introduction and Basics
Lecture -13

Today's Target

- Python version 2 and 3
- Boolean expression
- How memory is manage in python
- PEP 8

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

Difference-

Comparison Parameter	Python 2	Python 3
year of Release	Python 2 was released in the year 2000.	Python 3 was released in the year 2008.
“Print” Keyword	In Python 2, print is considered to be a statement and not a function.	In Python 3, print is considered to be a function and not a statement.
Storage of Strings	In Python 2, strings are stored as ASCII by default.	In Python 3, strings are stored as UNICODE by default.
Division of Integers	On the division of two integers, we get an integral value in Python 2. For instance, 7/2 yields 3 in Python 2.	On the division of two integers, we get a floating-point value in Python 3. For instance, 7/2 yields 3.5 in Python 3.
Exceptions	In Python 2, exceptions are enclosed in notations.	In Python 3, exceptions are enclosed in parentheses.

Difference-

Comparison Parameter	Python 2	Python 3
variable leakage	The values of global variables do change in Python 2 if they are used inside a for-loop.	The value of variables never changes in Python 3.
Iteration	In Python 2, the xrange() function has been defined for iterations.	In Python 3, the new Range() function was introduced to perform iterations.
Ease of Syntax	Python 2 has more complicated syntax than Python 3.	Python 3 has an easier syntax compared to Python 2.
Libraries	A lot of libraries of Python 2 are not forward compatible.	A lot of libraries are created in Python 3 to be strictly used with Python 3.
Usage in today's times	Python 2 is no longer in use since 2020.	Python 3 is more popular than Python 2 and is still in use in today's times.

Difference-

Comparison Parameter	Python 2	Python 3
Backward compatibility	Python 2 codes can be ported to Python 3 with a lot of effort.	Python 3 is not backward compatible with Python 2.
Application	Python 2 was mostly used to become a DevOps Engineer. It is no longer in use after 2020.	Python 3 is used in a lot of fields like Software Engineering, Data Science, etc.

Features

- **Open source and free**

- It is an open-source language which means that anyone can download it, use it, and share it. Moreover, it is free of cost.

- **Object-oriented**

- It supports object-oriented programming language features. For example, the concept of object and classes, encapsulation, inheritance, etc.

- **Extensible in nature**

- It is extensible in nature which means that we can use python code in other languages. For example C, C++ also can compile that code in C or C++.

Features

➢ **Integrated Language**

- We can easily integrate it with other languages such as C, C++, etc. Hence, it is an integrated language.

➢ **Interpreted Language**

- It uses an interpreter for converting the source code into machine code. This means that we execute the python code line by line. Hence, it becomes easy to debug the error and solve it.

➢ **Huge Standard Library**

- There are a very large number of libraries in python. These libraries contain predefined modules and functions for certain tasks. Hence, it becomes easy for the programmer to develop the code since he does not have to do all the things by himself. Moreover, the library is portable and cross-platform compatible.

BOOLEAN EXPRESSION

- Boolean expressions are the expressions that evaluate a condition and result in a Boolean value i.e true or false. Ex: (a>b and a> c) is a Boolean expression. It evaluates the condition by comparing if 'a' is greater than 'b' and also if 'a' is greater than 'c'. If both the conditions are true only then does the Boolean expression result is true. If any one condition is not true then the Boolean expression will result in false.
- Boolean expressions are written using Boolean operators (and) , (or) and (not)
- Example:

1. (x>1) and (x<5) - returns true if both the conditions are true, i.e if the value of 'x' is between 1 and 5.
2. (x%x) or (x%1) - returns true if any one condition is true, i.e: either 'x' is divisible by itself OR if 'x' is divisible by 1.
3. not (x==0) - returns true if the value of 'x' is other than 0. If the value of 'x' is 0 then it returns false

BOOLEAN EXPRESSION

```
x=10
z=(x>1) and (x<5)
 print(z)
```

Output

False

```
x=10
y=12
print(x>22 or y<25)
```

Output

true

BOOLEAN EXPRESSION

```
x=10
y=12
print(not (x>22 or y<25))
```

Output

false

PEP

PEP (Python Enhancement Proposal) is like a suggestion box for the Python programming language. It's a way for people who work on Python to suggest new ideas or changes to Python, and it helps everyone discuss, plan, and decide if those ideas are good for Python.

➢Just like when you have an idea, you write it down and share it with others, in the Python community, PEPs are used to write down and share ideas for making Python better. They explain what the idea is, why it's a good idea, and how it would work. Then, the Python community talks about it, and if most people agree it's a good idea, it might become a part of Python in the future.

➢In simple terms, a PEP is a way to suggest, discuss, and plan improvements or new features for Python, just like a suggestion box for your favorite programming language.

PEP

➢PEP stands for "Python Enhancement Proposal." PEPs are design documents that provide information to the Python community or describe a new feature or improvement in the Python programming language. They are the primary mechanism for proposing and discussing changes to Python's core language and standard library.

➢The PEP process is used to suggest and debate ideas, document the design and implementation of new features, and, in many cases, reach a consensus within the Python community before a change is accepted and incorporated into Python. PEPs cover a wide range of topics, from language enhancements to the development process itself

PEP

p

- PEPs are typically authored and submitted by Python developers and enthusiasts and go through a review and approval process. Some common types of PEPs include:
- PEP8 , PEP20, PEP257,PEP333 etc

PEP 8

➢PEP 8 is a Python Enhancement Proposal that outlines the style guide for writing clean and readable Python code. It provides a set of conventions and guidelines to make Python code consistent and easy to understand. Following PEP 8 recommendations is considered good practice and helps ensure that Python code is more accessible and maintainable.

➢Here are some key points from PEP 8 in plain language:

➢**Indentation**: Use 4 spaces for each level of indentation. Don't use tabs. Consistent indentation makes your code look neat and organized.

➢**Maximum Line Length**: Limit each line of code to a maximum of 79 characters for code, and 72 character

PEP 8

- **Whitespace**: Use a single space after commas, colons, and semicolons, but not immediately before. Avoid extraneous whitespace (e.g., at the end of lines).
- **Comments**: Write comments to explain complex or non-obvious parts of your code. Comments should be clear and concise. Use complete sentences and follow a consistent style for writing comments.
- **Naming Conventions**: Use descriptive variable and function names. Variable names should be lowercase with words separated by underscores (e.g., my_variable_name). Function names should be lowercase with words separated by underscores (e.g., my_function_name()).
- **Blank Lines**: Use blank lines to separate functions, classes, and blocks of code inside functions for better readability.

PEP 8

- **Whitespace in Expressions and Statements**: Avoid extraneous whitespace within expressions and statements. For example, don't put spaces immediately inside parentheses or square brackets.
- **Encoding**: Always specify the source code encoding as UTF-8 at the top of your Python files to ensure proper character encoding.
- **Documentation Strings (Docstrings)**: Write docstrings for modules, classes, and functions to explain their purpose and usage.
- **Imports**: Import statements should usually be on separate lines and should come at the top of your file. Imports should be grouped in the following order: standard library imports, related third-party imports, and local application/library specific imports.

PEP 8

➢Following PEP 8 helps maintain a consistent coding style across Python projects, making it easier for developers to collaborate and understand each other's code. It promotes clean, readable, and more maintainable Python code. Most Python developers adhere to PEP 8 recommendations to create code that is not only functional but also easy to work with.

How memory is managed in python

- In In Python, memory management, including the allocation and deallocation of memory on the private heap, is handled by the Python memory manager. Python uses a private heap to manage memory, and it employs a system of reference counting along with a cyclic garbage collector to keep track of and clean up memory when it's no longer in use.
- Here's how memory management works in Python:
- **Private Heap:** Python uses a private heap to store all its data structures and objects. The private heap is managed by the Python memory manager.
- **Reference Counting:** Each object in Python has a reference count associated with it. The reference count is the number of references or pointers to an object. When an object's reference count drops to zero, it means there are no references to that object, and it is considered to be eligible for deallocation

How memory is managed in python

- a=100
- b=100
- **python memory manager** creates only one object i.e "100" and reference count is "2". Whenever a new object is created pyhon manager will checks the memory for the object. If object already exists python manager will not create new object instead it references the name to the object and increases the reference counter of the object.
- Every python object holds three things
- object type
- object value
- object reference counter

How memory is managed in python

- Type int
- Value 100
- reference count 2
- reference a, b
- Example
- a = 100
- print(id(a)) # 10914336
- b = 100 print(id(b)) # 10914336
- print(id(a) == id(b)) # True

How memory is managed in python

- **Automatic Memory Allocation:** When you create a new object in Python (e.g., a variable, list, dictionary, etc.), the memory for that object is allocated from the private heap. Python takes care of this memory allocation automatically.
- **Memory Deallocation:** When the reference count of an object drops to zero (i.e., there are no more references to the object), Python's memory manager deallocates the memory associated with that object. This process is automatic and transparent to the programmer.

How memory is managed in python

➢**Garbage Collection:** In addition to reference counting, Python uses a cyclic garbage collector to detect and clean up objects with circular references (objects referencing each other in a cycle) that may not be detected by simple reference counting. This cyclic garbage collector identifies and reclaims memory for objects that are no longer reachable.

Garbage collection is the process of removing the unused variables/names and freeing the memory.

In python garbage collection is performed based on the reference count mechanism. Whenever the object is created the python will increases the reference count of the object by one. Whenever the reference is removed then it decreases the reference count by one. Garbage collector periodically checks and removes the objects from memory whose reference count is zero.

How memory is managed in python

- a = 100
- b = 100 # reference count of object 100 is 2
- del a # reference count of object 100 is 1
- del b # reference count of object 100 is 0
- # So, garbage collector will remove the object 100 from the memory..

How memory is managed in python

➢**Memory Leaks:** In some cases, if you create circular references or maintain references to objects unnecessarily, you can encounter memory leaks. These are situations where objects are not properly deallocated because their reference count never reaches zero. It's essential to be mindful of your code and ensure that you manage references appropriately to prevent memory leaks..

MODULE 1 : Introduction and Basics
Lecture -14

Today's Target

- Version of python
- Keywords and identifiers
- Print()

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

AKTU QUESTION

Q1	How memory is managed in python? Explain PEP8	AKTU 2019-20
Q2	Write a short note on Boolean expression	AKTU 2019-20
Q3	Explain the 5 benefits of using python	AKTU 2019-20
Q4	Discuss why python is interpreted language. Explain the history and features of python while comparing python 2 and python 3 version	AKTU 2022-23

Print()

. Python print() function prints the message to the screen or any other standard output device

syntax : print(value(s), sep= ' ', end = '\n', file=file, flush=flush)

Parameters:

value(s): Any value, and as many as you like. Will be converted to a string before printed

sep='separator' : (Optional) Specify how to separate the objects, if there is more than one.Default :' '

end='end': (Optional) Specify what to print at the end.Default : '\n'

file : (Optional) An object with a write method. Default :sys.stdout

flush : (Optional) A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False

Return Type: It returns output to the screen.

Print()

How print() works in Python?

You can pass variables, strings, numbers, or other data types as one or more parameters when using the print() function. Then, these parameters are represented as strings by their respective str() functions. To create a single output string, the transformed strings are concatenated with spaces between them.

. name = "Alice"

age = 25

print("Hello, my name is", name, "and I am", age, "years old.")

OUTPUT

Hello, my name is Alice and I am 25 years old.

Print()

➢**Python “end” parameter in print()**

➢The end keyword is used to specify the content that is to be printed at the end of the execution of the print() function. By default, it is set to “\n”, which leads to the change of line after the execution of print() statement.

➢. #This line will automatically add a new line before the# next print statement

➢print ("GATEWAY CLASS")

➢# This print() function ends with "**" as set in the end argument

➢print ("HELLO GUYS", end= "**")

➢print("BYE")

➢print("how are you", end="$$")

Print()

OUTPUT

GATEWAY CLASS

HELLO GUYSBYE**

how are you$$

Print()

By default, print() separates the items by spaces. The optional sep= parameter sets a different separator text

```
print(12, 24, -2, sep=':')

print('but', 'not', 'including', sep='**')

print('but', 'not', 'including', sep=' ')

print('hi','hello')

print("gateway","classes", sep='')
```

Output

```
12:24:-2

but**not**including

but not including

hi hello

gatewayclasses
```

Print()

flush parameter in Python with print() function

The I/Os in Python are generally buffered, meaning they are used in chunks. This is where flush comes in as it helps users to decide if they need the written content to be buffered or not. By default, it is set to false. If it is set to true, the output will be written as a sequence of characters one after the other. This process is slow simply because it is easier to write in chunks rather than writing one character at a time. To understand the use case of the flush argument in the print() function,

Version of python

version	Release date
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003

Version of python

version	Release date
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006
Python 2.6	October 1, 2008
Python 2.7	July 3, 2010
Python 3.0	December 3, 2008
Python 3.1	June 27, 2009
Python 3.2	February 20, 2011

Version of python

version	Release date
Python 3.3	September 29, 2012
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016
Python 3.7	June 27, 2018
Python 3.8	October 14, 2019
Python 3.3	September 29, 2012

p

keyword

➢**Python Keywords** are some predefined and reserved words in Python that have special meanings. Keywords are used to define the syntax of the coding. The keyword cannot be used as an identifier, function, or variable name. All the keywords in Python are written in lowercase except True and False

Rules for Keywords in Python

➢Python keywords cannot be used as identifiers.

➢All the keywords in Python should be in lowercase except True and False.

➢import keyword

➢print(keyword.kwlist)

keyword

- Keyword.kwlist
- This is a predefined variable of keyword module that stores all the keyword of python .Thus it can be used to display all the keyword of just by calling it
- Python provides an in-built module **keyword** that allows you to know about the reserved keywords of python.
- The keyword module allows you the functionality to know about the reserved words or keywords of Python and to check whether the value of a variable is a reserved word or not. In case you are unaware of all the keywords of Python you can use this module to retrieve this information. Also, it helps you to check whether a word is a keyword or not just by using its functions on Python shell mode

keyword

➢**keyword.iskeyword(parameter)**

This function returns **True** if the parameter passed is a Python keyword else returns **False**. The parameter can be a string or a variable storing a string. It accordingly compares the parameter with Python keywords defined in the language and returns the output.

keyword

- import keyword
- s="if“
- t="in“
- u="hello“
- print(s,"is a keyword in python",keyword.iskeyword(s))
- print(t,"is a keyword in python",keyword.iskeyword(t))
- print(u,"is a keyword in python",keyword.iskeyword(u))
- print("lambda is a keyword",keyword.iskeyword("lambda")
-)print("print is a keyword",keyword.iskeyword("print"))

p

keyword

- output
- if is a keyword in python True
- in is a keyword in python True
- hello is a keyword in python False
- lambda is a keyword True
- print is a keyword False

IMPORT

➢Import in python is similar to #include header_file in C/C++. Python modules can get access to code from another module by importing the file/function using import. The import statement is the most common way of invoking the import machinery, but it is not the only way

import module_name

When the import is used, it searches for the module initially in the local scope by calling __import__() function. The value returned by the function is then reflected in the output of the initial code.

```
import math
pie = math.pi
print("The value of pi is : ",pie)
```

➢The value of pi is : ', 3.14159265358979

IMPORT

➢In the above code module, math is imported, and its variables can be accessed by considering it to be a class and pi as its object.

The value of pi is returned by __import__(). pi as a whole can be imported into our initial code, rather than importing the whole module.

➢from math import pi

➢print(pi)

IMPORT

From module_name import *

In the below code module, math is not imported, rather just pi has been imported as a variable.

All the functions and constants can be imported using *.

```
from math import *
print(pi)
print(factorial(6))
```

```
 3.14159265359
720
```

Identifiers

Python Identifier is the name we give to identify a variable, function, class, module or other object.

UNIT-1: Introduction and basics

Lec-15

Today's Target

OUTPUT QUESTION

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

Output Question

```
x=6
y=3
print(x/y)
```

output

2.0

```
x=6
y=3
print(x//y)
```

output

2

```
x=5
y=4
print(x % y)
```

Output

1

```
x=5
y=7
print(x%y)
```

Output

5

```
x=5
y=4
x+=y
print(x)
x*=y
print(x)
```

Output

9

36

```
x-=y
print(x)
x/=y
print(x)
x%=y
print(x)
```

Output

32

8.0

0.0

Output Question

```
x=3
y=5
print(x==y)
print(x!=y)
```

Output

False

True

```
x=5
y=6
print(x>y)
print(x<y)
print(x>=y)
print(x<=y)
```

Output

False

True

False

True

```
x=True
y=False
print(x and y)
print(x or y)
print(not x)
print(not y)
```

Output

False

True

False

True

```
x=2*4+7
print(x)
y=2*(4+7)
print(y)
```

output

15

22

Output Question

```
x='24'+'45‘
print(x)
y=24+45
print(y)
```

Output

```
2445
69
```

```
x=input("enter a number")
print(type(x))
y=int(input("enter a integer"))
print(type(y))
z=float(input("enter a number"))
print(type(z))
```

Output

```
enter a number4
<class 'str'>
enter a integer4
<class 'int'>
enter a number4
<class 'float'>
```

Output Question

```
x=y=z=300
print("x =",x)
print("y=",y)
print("z=",z)
```

Output

```
x = 300
y= 300
z= 300
```

```
x,y,z=2,3,4
print("x=",x)
print("y=",y)
print("z=",z)
```

Output

```
x= 2
y= 3
z= 4
```

```
x=10*4+7-6/6
print(x)
```

output

```
46.0
```

Output Question

```
x=3/3=4/4//5*6
print(x)
```

Output

```
File "/home/main.py", line 1
    x=3/3=4/4//5*6
      ^^^
SyntaxError: cannot assign
to expression
```

```
x=3/34/4//5*6
print(x)
```

0.0

```
x=2*6**2
print(x)
```

Output

72

AKTU Full Courses (Paid)

Download Gateway Classes Application

From Google Play store

All Subjects

Link in Description

Thank You