**PYTHON**

**Day - 1**

1. **Introduction**
2. **Interpretor**
3. **Components of Python**
4. **Features**
5. **Applications of Python**
6. **Architecture**
7. **Disadvantages of Python**
8. **Python vs Java**
9. **Variables and Datatypes**
10. **Variable Scope**
11. **Identifiers**
12. **Number Types (Numeric Data Types)**
13. **Strings**
14. **Random Numbers**

**Introduction**

It is an interpreted, object oriented and high level programming language for general purpose.

**Interpreter**

Each statement translated into subroutines.

It executes one statement at a time.

It follows REPL philosophy. REPL – Read Evaluate Print Loop

**Components of Python**

**Functions**: Collection of statements and may return a value

**Classes**: Abstract data type

**Modules**: Collection of related classes and functions

**Packages**: Collection of related modules

**Features**

easy & simple to understand

interpretor

object oriented

free and open source

portable

GUI programming

large python library

**Applications of Python**

web applications

desktop applications (GUI)

automation (scripting and web browser automation)

robotics (enable tobotics)

web scraping (harvest data from website)

data science

artifical intelligence

game development

image processing (computer vision – face detection & color detection)

**Architecture**

**Parser –** user source code to generate abstract syntax tree

**Compiler –** abstract syntax tree to python bytecode

**Interpretor –** executes bytecode line by line

where,

**Abstract Syntax Tree –** way of representing programming syntax into hierarchical tree structure

**Disadvantages of Python**

speed limitation

weak in client side scripting

run time error (duck typing)

under developed database layers

**Python vs Java**

|  |  |
| --- | --- |
| **Python** | **Java** |
| slow (interpreted) | fast |
| easier than java | easy |
| dynamically typed (duck typing) | statically typed |
| data science, machine learning | embedded, cross platform applications |
| less number of lines | slightly high |
| semiclon not mandatory | semiclon mandatory |
| weak database layers | strong JDBC |
| intendation | braces |

**Variables and Datatypes**

**Variables**

Variable is a data name which is used to store the dta value.

Variable is a container for a value.

It can be assigned to a name.

It can be used it to refer later in the program.

Based on the assigned value, interpreter decides its data type.

**Datatypes**

Int

Float

Complex

String

Boolean

List

Tuple

Dictionary

Set

**Variable Scope**

**4 Types**

1. Local Scope
2. Non Local Scope (Enclosing Scope)
3. Global Scope
4. Built-In Scope

**Local Scope**

# localscope

a=1

def local():

    print(a)  #this variable “a” refers to the below “a” so it will throw error

    a=2

    print(a)

local()

Output : #UnboundLocalError: local variable 'a' referenced before assignment

def local(a=1):

    print(a) # this works fine as variable “a” defined in function definition

    a=2

    print(a)

local()

Output :

1

2

**Non Local Scope (Enclosing Scope)**

def red():

    a=1             # --> non local

    def blue():

        b=2         # --> local

        print(a)    # non local variable accessed here but can't edit this. to edit declare as nonlocal

        print(b)

    blue()

    print(a)

red()

Output :

1

2

1

def red():

    a=1

    def blue():

        nonlocal a

        a=2

        b=2

        print(a)

        print(b)

    blue()

    print(a)

red()

Output :

2

2

2

**Global Scope**

a=1

def counter():

    a=2

    print(a) #this refers local variable "a" to affect this -> declare variable "a" as global

counter()

print(a)

Output :

2

1

a=1

def counter():

    global a

    a=2

    print(a)

counter()

print(a)

Output :

2

2

**Built-In Scope**

This has wider scope in python as this has all the names that are loaded into python

**Ex:**

print()

id()

type()

**Ex1:**

print(id(2))

identity = id       #here we associate the name identity with built-in function id()

print(identity(2))

Output:

1568293519632

1568293519632

**Ex2:**

def sayhello(): print("Hello")

hi=sayhello()

print(hi)

print(type(hi))

Output:

Hello

None

<class 'NoneType'>

Since the sayhello function does not return anything, we get object type as None.

None means no value.

**Ex3:**

def sayhello():

    print("Hello")

    return 21

hi=sayhello()

print(hi)

print(type(hi))

Output:

Hello

21

<class 'int'>

**Numeric Data Types**

No limitation or length for int value.

>>> a=102021

>>> type(a)

<class 'int'>

>>> isinstance(a,int)

True

>>>

|  |  |
| --- | --- |
| **Number System** | **Prefix** |
| Binary | 0b (or) 0B |
| Octal (0-7) | 0o (or) 0O |
| Hexadecimal (0-9, A-F) | 0x (or) 0X |

>>> print(0b111)

7

>>>

Can not convert complex type into int, float, bin, hexadecimal and octal

>>> (1.1+2.2)==3.3

False

>>> 1.1+2.2

3.3000000000000003 #this is the reason for above false

>>>

>>> from fractions import Fraction

>>> Fraction(1.5)

Fraction(3, 2)

>>>

**Strings**

Sequence of characters

**Formats**

%d – integer

%s – String

%f – Float

**String Formats**

There are 3 ways

**1st Way**

use “%” for string formats

# 1st way - use %

x=10

printer="Dell"

print("Print %s pages from %s printer"%(x,printer))

Output:

Print 10 pages from Dell printer

**2nd Way**

use .format method

# 2nd way

print("Print {0} pages from {1} printer".format(x,printer))

print("Print {page} pages from {printer\_type} printer".format(page=21,printer\_type="Canon"))

Output:

Print 10 pages from Dell printer

Print 21 pages from Canon printer

**3rd Way**

use “f” format method

# 3rd way

print(f"Print {x} pages from {printer} printer")

Output:

Print 10 pages from Dell printer

>>> "hey"<"hi"

True # reason for true is i comes after e in alphabet

>>>

>>> a="check"

>>> a=="check"

True

>>>

>>> 'yes' != 'no'

True

>>>

>>> 'ba'+'na'\*2

'banana'

>>>

>>> ' ' and '1'

'1'

>>> '' and '1'

''

>>> '1' and 'out'

'out'

>>>

**String Methods**

**find (returns index)**

>>> "hello".find('e')

1

>>> "hello".find('r')

-1 # -1 for not available value

>>>

**replace**

>>> "banana".replace("na","la")

'balala'

>>>

**split**

>>> "I,am,ok,now".split(',')

['I', 'am', 'ok', 'now']

>>> "Get.Lost.From.Here".split('.')

['Get', 'Lost', 'From', 'Here']

>>>

**join**

>>> "\*\*".join(['hi','hello','bye'])

'hi\*\*hello\*\*bye'

>>>

**capitalize**

>>> "hi hello weclome".capitalize()

'Hi hello weclome'

>>>

**center**

>>> "hi hello weclome".center(31,'\*')

'\*\*\*\*\*\*\*\*hi hello weclome\*\*\*\*\*\*\*'

>>>

**Len**

>>> len('\*\*\*\*\*\*\*\*hi hello weclome\*\*\*\*\*\*\*')

31

>>>

**count**

>>> '\*\*\*\*\*\*\*\*hi hello weclome\*\*\*\*\*\*\*'.count('\*')

15

>>>

**encode**

>>> "python".encode(encoding='utf-8',errors='strict')

b'python'

>>> 'pythön'.encode(encoding='utf-8',errors='strict')

b'pyth\xc3\xb6n'

>>>

**endswith**

>>> "hi hello weclome".endswith('come')

False

>>> "hi hello weclcome".endswith('come')

True

>>>

**index**

>>> "hi hello weclcome".index('e')

4

>>> "hi hello weclcome".index('r') # Output: ValueError: substring not found

>>>

**isalnum**

>>> "123456789hello".isalnum()

True

>>>

**isalpha**

>>> "hello".isalpha()

True

>>>

**isdecimal**

>>> "2021".isdecimal()

True

>>>

**isdigit**

>>> "2021".isdigit()

True

>>>

**isidentifier**

>>> "try".isidentifier()

True

>>>

**islower**

>>> "bye".islower()

True

>>>

**isupper**

>>> "HELLO".isupper()

True

>>>

**isspace**

>>> " ".isspace()

True

>>>

**istitle**

>>> "Hello World".istitle()

True

>>>

**swapcase**

>>> "hi HELLO bye".swapcase()

'HI hello BYE'

>>>

**title**

>>> "hi hello bye".title()

'Hi Hello Bye'

>>>

**strip**

strip function will remove the whitespace only before and after the string but it will not remove the space (or given option) in between the string

>>> a=" hi hello welcome "

>>> a.strip()

'hi hello welcome'

>>>

**Random Numbers**

**Generate 4 digits random number without using random function**

import time

for \_ in range(4):

    time.sleep(0.2)

    print(int((time.time()\*1000)%10),end='')

Output:

7889

**Generate random value using random module**

**random()** - generates a floating-point number between 0 and 1

from random import random

print(random())

Output:

0.12993059894243253

**randint()** - generates an integer number between the given range

**Ex:** (0-9)

from random import randint

print(randint(0,9))

Output:

7

**randrange()** - generates a random number from the given range

from random import randrange

print(randrange(1,10,2))

#syntax: randrange(start, stop, step)

Output:

5

**sample()** - generates unique random numbers within the given range of numbers

from random import sample

print(sample(range(1,51),15))

Output:

[50, 8, 44, 9, 43, 15, 6, 11, 16, 19, 23, 18, 7, 45, 17]

**choice()** – picks a random value from our own custom list

from random import choice

lst=['hi','hello','bye']

print(choice(lst))

Output:

hi

**choices()** – picks multiple random value from our own custom list

from random import choices

lst=['hi','hello','welcome','bye']

print(choices(lst,k=3)) #k=3, choose 3 randomw values from list nut not unique values(may duplciate occurs)

Output:

['bye', 'hello', 'welcome']

**shuffle()** – shuffling a list randomly

from random import shuffle

lst=['hi','hello','welcome','bye']

shuffle(lst)

print(lst) #print list after shuffle

Output:

['hello', 'bye', 'hi', 'welcome']

**seed()**

It helps to generate the same random value again and again.

It is one of the ways of controlling random number generator.

If there is no seed(n) defined, current system time will be taken for generating random numbers.

from random import seed, randint

seed(21)

print(randint(0,9),randint(0,9),randint(0,9),randint(0,9))

Output:

2 6 6 4

**rand()** – Generate random number using NumPy

from numpy.random import rand

print(rand(3))

Output:

[0.45979486 0.20758488 0.19856811]

**Day - 2**

1. **Operators**
2. **Bitwise Operators**
3. **Comparision Operators**
4. **Operator Overloading**
5. **Ternary Operator**
6. **Operator Precedence**
7. **Namespaces**
8. **Decision Making**
9. **Switch Case**
10. **Python Data Structure**

**Operators**

There are many operators in python

**Arithmetic Operators**

>>> a=5

>>> b=9

>>> a+b

14

>>> a-b

-4

>>> a\*b

45

>>> a/b

0.5555555555555556

>>> b%a

4

>>> b\*\*a

59049

>>> b//a

1

>>>

**Relational Operators**

>>> a=5

>>> b=9

>>> a>b

False

>>> a<b

True

>>> a>=b

False

>>> a<=b

True

>>> a==b

False

>>> a!=b

True

>>>

**Assigment Operators**

>>> a=5

>>> b=2

>>> a+=b

>>> a

7

>>> a-=b

>>> a

5

>>> a\*=b

>>> a

10

>>> a/=b

>>> a

5.0

>>> a%=b

>>> a

1.0

>>>a\*\*=b

>>> a

1.0

>>> a//=b

>>> a

0.0

>>>

**Logical Operators**

>>> a=5

>>> b=2

>>> a and b

2

>>> a or b

5

>>> True and False

False

>>> True or False

True

>>> not(5)

False

>>> not(False)

True

>>>

**Membership Operators**

>>> 'hello' in ['hi','hello','bye']

True

>>> 'welcome' in ['hi','hello','bye']

False

>>> 'welcome' not in ['hi','hello','bye']

True

>>>

**Identity Operators**

>>> 20 is 21

False

>>> 2 is 2

True

>>> 2 is '2'

False

>>> 2 is '2.0'

False

>>> 2 is 2.0

False

>>> 2 is not '2'

True

>>>

**Bitwise Operators**

**Bitwise AND (&)**

|  |  |
| --- | --- |
| **0 & 0** | **0** |
| **0 & 1** | **0** |
| **1 & 0** | **0** |
| **1 & 1** | **1** |

>>> True & False

False

>>> '$' & 'abc'

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for &: 'str' and 'str'

>>>

**Bitwise OR (|)**

|  |  |
| --- | --- |
| **0 | 0** | **0** |
| **0 | 1** | **1** |
| **1 | 0** | **1** |
| **1 | 1** | **1** |

>>> True | False

True

>>> '$' | 'abc'

TypeError: unsupported operand type(s) for |: 'str' and 'str'

>>>

**Bitwise XOR (^)**

|  |  |
| --- | --- |
| **0 ^ 0** | **0** |
| **0 ^ 1** | **1** |
| **1 ^ 0** | **1** |
| **1 ^ 1** | **0** |

>>> True ^ False

True

>>> True ^ True

False

>>> '$' ^ 'abc'

TypeError: unsupported operand type(s) for ^: 'str' and 'str'

>>>

**Bitwise 1’s Complement**

>>> ~2

-3

>>> bin(2)

'0b10'

>>> bin(-3)

'-0b11'

>>>

**Bitwise Left Shift Operator (<<)**

Add 0s to the empty least significant places

>>> True<<2

4

>>> 2<<1

4

>>> 3<<2

12

>>>

**Bitwise Left Shift Operator (<<)**

Bits will be lost here where as bits will be added in shift left operator

>>> 3>>1

1

>>> 31>>3

3

>>>

**Comparison Operators**

>>> 3>3

False

>>> 3<3

False

>>> 3>=3

True

>>> 3<=3

True

>>> 'Abc'<'abc'

True # because of Ascii value A-65 a-97 so 65<97

>>> 0.999<1

True

>>> (1,2,3)<(1,2,3,4)

True

>>> (1,3,2)<(1,2,3)

False

>>> (1,2,3)<(1,3,2)

True

>>> () < (0,)

True

>>> (1,2) < ('one','two')

TypeError: '<' not supported between instances of 'int' and 'str'

>>> [0]<[False]

False

>>> (1,'one')<(2,'two')

True

>>> {1,2,3}<{1,3,2}

False # because set will rearrange its values as {1,2,3} so it returns False

>>> {1:'one',2:'two'}<{1:'three',2:'four'}

TypeError: '<' not supported between instances of 'dict' and 'dict'

>>> 0.5<False

False

>>> 3,4,5>3,4,5.0

(3, 4, True, 4, 5.0)

>>> 3,4,5 > 3,4,5.0

(3, 4, True, 4, 5.0)

>>> 3,4,5>(3,4,5.0)

TypeError: '>' not supported between instances of 'int' and 'tuple'

>>> (3,4,5)>(3,4,5.0)

False

>>> 3=='3'

False

>>> {1,2,3}=={1,3,2}

True

>>> 0==False

True

>>> 3!=3.0

False

>>> 3==3.0

True

>>>

**Operator Overloading**

Same operator for different purpose

>>> 42+1

43

>>> '42'+'1'

'421'

>>> 'hello'+' '+'world'

'hello world'

>>> [1,2,3]+[4,5,6]

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

>>> (1,2,3)+(4,5,6)

(1, 2, 3, 4, 5, 6)

>>>

**Ternary Operator**

It is also known as conditional operator

**Syntax: [true] if [expression] else [false]**

>>> a=5

>>> b=9

>>> result = a if a>b else b

>>> result

9

>>>

There is an another way to achieve this

>>> a=5

>>> b=9

>>> res = (b,a)[a>b]

>>> res

9

>>>

In the above code

* (9,5)[5>9]
* (9,5)[0]
* Ans: 9

**Nested Ternary Operator**

from random import random

a=random()

print("less than 0" if a<0 else "between 0 and 1" if a>=0 and a<=1 else "greater than 1")

Output:

between 0 and 1

**Switch Case**

Python does not have in-built switch case option. But this can be achieved in other ways.

def week(i):

    switcher={

        0:'sunday',

        1:'monday',

        2:'tuesday',

        3:'wednesday',

        4:'thursday',

        5:'friday',

        6:'saturday'

    }

    return switcher.get(i,"Invalid day of week")

print(week(4))

def zero():

    return 'zero'

def one():

    return 'one'

def indirect(i):

    switcher={

        0:zero,

        1:one,

        2:lambda:'two'

    }

    func=switcher.get(i,lambda:'Invalid')

    return func()

print(indirect(2))

print(indirect(5))

**Python Data Structures**

List

Tuple

Set

Dictionary