13 June 2020

15:00

Python Scripting

section1            1,2-june

section2            3-june

section3            4,5-june

section4            6,7-june

section5            8,9-june

section6            10,11-june

section7            12-june

section8            13,14-june

section9            15-june

section10           16,17-june

section11           18,19,20,21,22-june

section12           23-june

section13           29,30-june

section14           1,2-july

section15           2,3-july

section16           4-july

section17           5,6-july

section18           7,8,9,10,11-july

section19           12,13,14,15-july

section20           16-july

section21           17-july

section22           18,19,20,21,22-july

section23           23-july

========================================

python            =3 hours (6:30-8:30,9:30-10:30)

AWS Associate    =4hours     (10:30-12:30,2:30,4:30)

Docker            =4 hours (4:30-5:30,6:30-8:30,9:30-10:30)

========================================

**python:-**

indent is knows as space

indentations are used to represent a block of code

don't provide spaces/indent unnecessarily

[root@naveenhost python]# cat indent.py

print("This is about indentation")

if 3 > 1:

        print("using if condition")

        print("we are comparing 3 and 1")

[root@naveenhost python]# python indent.py

This is about indentation

using if condition

we are comparing 3 and 1

[root@naveenhost python]#

**special characters:-**

Always write special characaters inside quotes

/n new line

/b back quotes

>>> x=5.6

>>> print(x)

5.6

>>> print(type(x))

<class 'float'>

>>> y=6

>>> print(y)

6

>>> print(type(y))

<class 'int'>

**variables:-**

check in /root/python/section3/variables.py

**DataTypes:-**

alphabets cannot be converted into numbers and

numbers can be converted into alphabets

int

float

complex

str

bool

Note:- Any data can be converted into boolean

----

       bool(any\_data\_type)=True or False

       bool(empty)=False bool(0),bool(None),bool([]),bool({}),bool(())

       bool(non-empty)=True

check in /root/python/section3/data\_types.py

**working with multiple variables and strings in print:-**

check in /root/python/section3/multiple\_variables\_in\_print.py

#!/usr/local/bin/python3

"""

x=3;y=5.7;my\_name="python scripting"

"""

x=3

y=5.7

my\_name="python scripting"

#print(x,y,my\_name)

print("{}\n{}\n{}".format(x,y,my\_name))

print("{}\n{}\n{}".format(y,x,my\_name))

my\_req\_output=f"x value is: {x}\ny value is: {y}\nmy\_name value is: {my\_name}"

print(my\_req\_output)

**Input and output syntax:-**

check in /python/section3/input\_and\_output.py

**join center and zfill string operations:-**

check in /root/python/section4/join.py

**strip and split operations:-**

check in /root/python/section4/strip\_split\_operations.py

**count index and find operations on strings:-**

>>> x="python is easy and it is scripting"

>>> print(x.count('i'))

5

>>> print(x.count('is'))

2

>>> print(x.count('it'))

1

>>> print(x.count('a'))

2

>>> print(x.count('m'))

0

>>> print(x.count('f'))

0

>>> print(x.count('s'))

4

>>> print(x.index('p',1))

29

>>> print(x.index('p',29))

29

>>> print(x.index('p',30))

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

ValueError: substring not found

>>> print(x.index('is'))

7

>>> print(x.index('is',8))

22

>>> print(x.index('is',10))

22

>>> print(x.index('is',23))

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

ValueError: substring not found

>>>

**Display string at right/left/center of a line in title format:-**

>>> import os

>>> os.get\_terminal\_size()

os.terminal\_size(columns=70, lines=44)

>>> os.get\_terminal\_size().columns

70

>>>

check in /root/python/section4/string\_title.py

**Data Structures of Python:-**

Data Structures are used to store a collection of data

There are four built-in data structures

They are:

    List []

    Tuple()

    Dictionary {} with key value pair

    set {}

>>> x=5

>>> print(x)

5

>>> my\_values=[3,4,5,'python']

>>> print((my\_values[1]))

4

>>> print((my\_values)[1])

4

>>>

**List Data Structures of Python:-**

lists are mutable(we can change)

when we try to to bool for empty list it gives output as below

>>> my\_list=[]

>>> print(bool(my\_list))

False

>>> my\_list=[2,3,4,'python']

>>> print(my\_list)

[2, 3, 4, 'python']

>>> print(bool(my\_list))

True

>>>

>>> my\_list=[3,2,4,'python',5.6]

>>> print(my\_list)

[3, 2, 4, 'python', 5.6]

>>> print((my\_list)[3])

python

>>> print((my\_list)[4])

5.6

>>> print(type(my\_list))

<class 'list'>

>>>

**Dictionaries:-**

Dictonaries are represented with {}

check in /python/section5/dictionary\_data\_structure.py dictionary\_data\_structure1.py

**Sets:-**

- sets are called unordered collection of data

- if we give duplicates of data and sets will remove

  duplicates and provides unique data

my\_sets=set({})

print(type(my\_sets))

print(bool(my\_sets))

>>> my\_list=[2,4,6,7,0,7,12]

>>> print(set(my\_list))

{0, 2, 4, 6, 7, 12}

>>> dir(set)

['\_\_and\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_',

'\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_',

'\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iand\_\_',

'\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_ior\_\_', '\_\_isub\_\_',

'\_\_iter\_\_', '\_\_ixor\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_',

'\_\_ne\_\_', '\_\_new\_\_', '\_\_or\_\_', '\_\_rand\_\_', '\_\_reduce\_\_',

'\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_ror\_\_', '\_\_rsub\_\_',

'\_\_rxor\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_',

'\_\_sub\_\_', '\_\_subclasshook\_\_', '\_\_xor\_\_', 'add', 'clear',

'copy', 'difference', 'difference\_update', 'discard',

'intersection', 'intersection\_update', 'isdisjoint',

'issubset', 'issuperset', 'pop', 'remove',

'symmetric\_difference', 'symmetric\_difference\_update',

'union', 'update']

>>> a={3,4,5,6}

>>> b={7,8,2,3,4}

>>> a.union(b)

{2, 3, 4, 5, 6, 7, 8}

>>> a.intersection(b)

{3, 4}

>>>

**Introduction to python operators:-**

- operators are the important concept in any language

- The operator can be defined as a symbol which is responsible

  for a particular operation between two operands

- Example + is an operator to perform addition operation on operands

- Operands may be values or variables or combination of

  values and variables

- Python provides a variety of operators described as follows

  - Arithmetic operators (Takes values as inputs,performs its

    operation on input values and gives output as values)

  - Assignment operators (Takes values as inputs,performs its

    operation on input values and gives values as outputs)

  - Comparision operators (Takes values as inputs,performs its

    operation on input values and gives output as either True or False)

  - Identity operators (Takes values as inputs,performs its

    operation on input values and gives output as either True or False)

  - Membership operators (Takes values as inputs,performs its

    operation on input values and gives output as either True or False)

  - Logical operators (Takes True or False as inputs,performs

    its operation on this inputs and give output as either True or False)

  - Bitwise operators (Takes values as inputs,performs

    operations on its binary representation and gives output as a value)

**Arithmetic operators:-**

>>> 2+2

4

>>> c=2+2

>>> print(c)

4

>>> c

4

>>> c=2\*2

>>> c

4

>>> print(c)

4

>>> c=4/8

>>> c

0.5

>>> c=8/4

>>> c

2.0

>>> c=9/4

>>> c

2.25

>>> 3\*\*2

9

>>> a=5

>>> b=3

>>> a\*\*b

125

>>> result=a\*\*b

>>> print(result)

125

>>> c=17%3

>>> print(c)

2

>>> 8//2

4

>>> 7//2

3

>>> 6//2

3

>>> 15//2

7

>>> 17/3

5.666666666666667

>>> 17//3

5

>>> 29//3

9

>>>

**Assignment operator:-**

>>> a=3

>>> a=a\*10+2

>>> a

32

>>> a=3

>>> b=2

>>> a+=b

>>> a

5

>>> a=a+b

>>> a

7

>>> a=2

>>> b=3

>>> a+=b

>>> a

5

>>>

operator        Example            same as

=                a=b             a=b

+=                a+=b            a=a+b

-=                a-=b            a=a-b

\*=                a\*=b            a=a\*b

/=                a/=b            a=a/b

%=                a%=b            a=a%b

**Comparision operator:-**

> < == != >= <=

>>> 3 > 4

False

>>> 3 < 4

True

>>> 4 = 4

  File "<stdin>", line 1

SyntaxError: can't assign to literal

>>> 4 == 4

True

>>> 3 >= 4

False

>>> a=4

>>> 3 > a

False

>>> 'a' == 'b'

False

>>> 'a' == 'a'

True

>>> 'a' < 'b'

True

>>> ord('a')

97

>>> ord('b')

98

>>> ord('-')

45

>>> chr(45)

'-'

>>> 'aaa' < 'acb'

True

>>> 'aac' > 'abc'

False

>>>

**identify and membership operators:-**

**identity operator:-**

- identity operators are used to find the type of: class/type/object

- They are two types of identity operators

    - is

    - is not

>>> x=6

>>> x

6

>>> y='hi'

>>> type(x)

<class 'int'>

>>> type(y)

<class 'str'>

>>> x==y

False

>>> x=6

>>> y=6

>>> x==y

True

>>> type(x)

<class 'int'>

>>> type(y)

<class 'int'>

>>> type(x) is type(y)

True

>>> z='hi'

>>> type(x) is type(z)

False

>>> type(x) is not type(z)

True

>>>

**Membership operator:-**

* + if something that you required is there are not in given list we are using membership operators
  + Membership operators are used to validate the membership of a value

        - in

        - not in

>>> x=[4,5,6,7]

>>> 6 in x

True

>>> 90 in x

False

>>> valid=['1.6','1.7','1.8','1.9']

>>> host\_java="1.5"

>>> host\_java in valid

False

>>>

check in /python/section6/membership\_operator.py

**Logical operators:-**

* + Logical operators are useful to combine multiple conditions
  + They are three type of logical operators in python
  + They are:  
         - and  
         - or  
         - not
  + For AND operator - it returns TRUE if both the operands(right side and left side) are true
  + For OR operator - it returns TRUE if either of the operand(right side or left side) is TRUE
  + For NOT operator - not True equal to False and not False equal to True

>>> 3 > 1

True

>>> 1 in [3,4,5]

False

>>> 3 > 1 in [3,2,1]

True

>>> 3 > 1 and 1 in [3,4,5]

Faxlse

>>> 3 > 1 and 1 in [1,2,3]

True

>>> 3 < 1 and 1 in [1,2,3]

False

>>> 1 < 2 and 2 < 4 and 5 < 6

True

>>> 1 < 2 and 2 < 4 and 5 > 6

False

>>> 1 < 2 or 3 < 4

True

>>> 1 < 2 or 4 < 3

True

>>> 1 < 2

True

>>> not 1 < 2

False

>>> 2 < 1

False

>>> not 2 < 1

True

>>> all([2 < 3,4 < 5,5 < 6,7 < 10]) (all is same like and operator)

True

>>> any([2 > 4,4 > 2]) (any is same like OR operator)

True

>>> not any([2 > 4,4 > 2])

False

>>> 3 < 1 or 4 < 1 (False or False then output False)

False

>>> 3 > 1 or 4 > 1 (True or true then output True)

True

>>>

**Conditional statements of python:-**

* + IF is called simple conditional statement
  + IF is used to control the execution of set of lines or block of code or one line

if expression:

            statement1

            statement2

comparision operators

identity operators

membership operators

logical operators

check in /python/section6/case.py if\_numbers.py

**if else conditional statement:-**

check in /python/section7/get\_title\_format.py find\_greater\_number.py

**practice with conditional statement:-**

check in /python/section7/number\_to\_word.py

**Introduction to python module:-**

A module is a file containing python definitions and statements,That means, module containing python functions,classes and variables

**use of module:-**

* + Reusability

if script name is my\_module.py then module name is my\_module

**Types of python modules:-**

* + Default modules
  + Third party modules

import either default or third party modules before using them

**Different ways to import third party modules:-**

* + using import math
  + using from:

                from math import \* (\* means all the operations)

by using modules we can reduce the length of the code

>>> import math

>>> print(math.pi)

3.141592653589793

>>> print(math.pow(2,3))

8.0

>>>

>>> help("modules")

>>> import math

>>> dir(math)

['\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']

>>> math.pow(2,3)

8.0

>>> math.pi

3.141592653589793

>>> help(math)

>>> import csv

>>> dir(csv)

['Dialect', 'DictReader', 'DictWriter', 'Error', 'OrderedDict', 'QUOTE\_ALL', 'QUOTE\_MINIMAL', 'QUOTE\_NONE', 'QUOTE\_NONNUMERIC', 'Sniffer', 'StringIO', '\_Dialect', '\_\_all\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', '\_\_version\_\_', 'excel', 'excel\_tab', 'field\_size\_limit', 'get\_dialect', 'list\_dialects', 're', 'reader', 'register\_dialect', 'unix\_dialect', 'unregister\_dialect', 'writer']

>>> import xlwt

>>> dir(xlwt

... )

['ANTLRException', 'Alignment', 'BIFFRecords', 'Bitmap', 'Borders', 'Cell', 'Column', 'ExcelFormula', 'ExcelFormulaLexer', 'ExcelFormulaParser', 'ExcelMagic', 'Font', 'Formatting', 'Formula', 'Pattern', 'Protection', 'Row', 'Style', 'UnicodeUtils', 'Utils', 'Workbook', 'Worksheet', 'XFStyle', '\_\_VERSION\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_path\_\_', '\_\_spec\_\_', 'add\_palette\_colour', 'antlr', 'compat', 'easyfont', 'easyxf', 'struct']

**platform module:-**

The platform module is used to access the underlying platforms data such as hardware, operating system and interpreter version information

**How to use platform module in a script:-**

First import platform module using below syntax:

    import platform

    import platform as pt

    from platform import \*

    from platform import system,platform

**How to list all functions and variables of a platform module:-**

There is abuilt-in function names(or variable names) in a module and that is dir() function

Then use dir() function as:

        print(dir(platform))

>>> dir(platform)

>>> len(dir(platform))

72

>>>help(platform)

>>> import platform

>>> print(platform.machine())

x86\_64

>>> print(platform.release())

3.10.0-1062.12.1.el7.x86\_64

>>> print(platform.platform())

Linux-3.10.0-1062.12.1.el7.x86\_64-x86\_64-with-centos-7.7.1908-Core

>>> print(platform.architecture())

('64bit', 'ELF')

>>> print(platform.processor())

x86\_64

>>> print(platform.node())

[naveenhost.example.com](http://naveenhost.example.com/)

>>> print(platform.uname())

uname\_result(system='Linux', node='[naveenhost.example.com](http://naveenhost.example.com/)', release='3.10.0-1062.12.1.el7.x86\_64', version='#1 SMP Tue Feb 4 23:02:59 UTC 2020', machine='x86\_64', processor='x86\_64')

check in /python/section8/working\_with\_platform\_module.py

**getpass module:-**

getpass() getuser()

**getpass:-**

* + getpass() prompts the user for a password without echoing
  + The getpass module provides a secure way to handle the password prompts where programs interact with the users via the terminal

check in /root/python/section8/working\_with\_getpass\_module.py

**getuser:-**

* + getuser function displays the login name of the user
  + This function checks the environment variables LOGNAME,USER,LNAME and USERNAME,in order, and returns the value of the first non-empty string

>>> import getpass

>>> getpass.getuser()

'root'

>>> ^C

KeyboardInterrupt

>>>

[root@naveenhost python]# env | grep -i user

USER=root

XDG\_RUNTIME\_DIR=/run/user/0

[root@naveenhost python]# env | grep -i logname

LOGNAME=root

**sys module:-**

* + The sys module is used to work with python runtime environment
  + The sys module provides functions and variables used to manipulate different parts of the python runtime environment

import sys

dir(sys)

help(sys)

check in /python/section9/sys\_module.py

**sys.argv of sys modules:-**

* + sys.argv returns a list of command line arguments passed to a python script

check in /python/section9/command\_line\_arguments.py sys\_module.py working\_with\_strings.py

**os module:-**

* + This module is used to work/interact with operating system to automate many more tasks like creating directory,removing directory,identifying current directory and many more

Os.sep

Os.getcwd()

Os.chdir(path)

Os.listdir()

Os.listdir(path)

Os.mkdir(path)

Os.makedirs(path) (Recursive directory creation function)

Os.remove(path)

Os.removedirs(path) (Remove directories recursively)

Os.rmdir(path)

Os.rename(src,dst)

Os.environ()

Os.getuid()

Os.getpid()

>>> import os

>>> dir(os)

>>> print(os.sep)

/

>>> import os

>>> os.chdir("/root/python/section10/")

>>> print(os.getcwd())

/root/python/section10

>>> import os

>>> print(os.listdir())

['section2', 'section3', 'section4', 'section5', 'section6', 'section7', '.git', 'section8', 'section9', 'section10']

>>> os.chdir("/root/python/section9")

>>> print(os.getcwd())

/root/python/section9

>>> print(os.listdir())

['sys\_module.py', 'command\_line\_arguments.py', 'working\_with\_strings.py']

>>> print(os.getcwd())

/root/python/section9

>>> print

<built-in function print>

>>> print(os.listdir())

['sys\_module.py', 'command\_line\_arguments.py', 'working\_with\_strings.py']

>>> os.chdir("/root/python/section10/")

>>> print(os.getcwd())

/root/python/section10

>>> os.mkdir("module")

>>> print(os.listdir())

['module']

>>> print(os.getcwd())

/root/python/section10

>>> os.chdir("/root/python/section10/")

>>> print(os.getcwd())

/root/python/section10

>>> os.mkdir("module")

>>> print(os.listdir())

['module']

>>> print(os.getcwd())

/root/python/section10

>>> os.makedirs("modules1/module/mod")

>>> print(os.lisdir())

Traceback (most recent call last):

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

AttributeError: module 'os' has no attribute 'lisdir'

>>> print(os.listdir())

['module', 'modules1']

>>> print(os.chdir("/root/python/section10/modules1/module/mod/"))

>>>print(os.environ)

>>> print(os.getuid())

0

>>> print(os.getpid())

22042

>>> print(os.getgid())

0

>>>

**os.path module:-**

os.path is a sub module of os module

os.path.sep

os.path.basename(path)

os.path.dirname(path)

os.path.join(path1,path2)

os.path.split(path)

os.path.getsize(path)

os.path.exists(path)

os.path.isfile(path)

os.path.isdir(path)

os.path.islink(path)

After time module we will also discuss: getatime,getctime and getmtime

check in /python/section10/working\_with\_os\_path.py

**os.system() from os module:-**

os.system() is a function

>>> import os

>>> os.getcwd()

'/root/python/section10'

>>> os.listdir()

['working\_with\_os\_path.py']

>>> os.system("cat working\_with\_os\_path.py")

>>> os.system("dir")

working\_with\_os\_path.py

0

>>> os.system("c")

sh: c: command not found

32512

>>> os.system("ls")

working\_with\_os\_path.py

0

>>> rt=os.system("ls")

working\_with\_os\_path.py

>>> print(rt)

0

>>>

check in /python/section10/working\_with\_os\_system.py

platform\_os\_module.py

**os.walk(path):-**

it is used to generate directory and file names in a directory tree by walking

loops:-