OSTAD (DATA SCIENTIST JOURNEY)

Class-1(Note)

PYTHON

Python Comments

Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

Single line Comment

Comments starts with a `#`, and Python will ignore them:

```python
comments.py


print("Hello world") #This is a Comment

```

Multiline Comments

Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code using """ """

```python

print("Hello world")

"""
This is multiline comment
so you have to careful.

"""
```

Python Variables

Variables are containers for storing data values.

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

1.

```
x=10  #variable
print(x)
```

2.

If you assign both an integer and a string to the same variable name, like `x = 4` and then `x = "hello string"`, the variable `x` will be updated to the last assigned value, and that value will be printed.

```
x=10  #variable
x="Hello ostad"
print(x)

#   'Hello world' will be printed.
```

Single or Double Quotes?

String variables can be declared either by using single or double quote.

```
p="Hello Disha"  #Double quote
q='Hello Disha'  #single quote(is the same as before)

print(p,q)

#output:
#  Hello Disha Hello Disha
```

3.

But if we print
`x` before updating it with the new value, the output will first show the initial value. Then, after the update, it will show the new value.

```
x=10  #variable
print(x)
x="Hello ostad"
print(x)

#output:
# 10
# Hello ostad
```

Python Type Casting

Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)

- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

1.integers

```python
x=int(1) #x will be 1
y=int(2.8) #y will be 2
z=int("3") #z will be 3

print(x,y,z)
```

2.floats

```python
x=float(1) #x will be 1.0
y=float(2.8) #y will be 2.8
z=float("3") #z will be 3.0
w=float("4.2") #w will be 4.2

print(x,y,z,w)
```

3.strings

```python
x=str("s1") #x will be s1
y=str(2) #y will be 2
z=str("3.0") #z will be 3.0


print(x,y,z)
```

Type casting in python(implicit and explicit)

two types of Type Casting in Python:

- Python Implicit Type Conversion
- Python Explicit Type Conversion

1.Implicit Type conversion

Python converts the datatype into another datatype automatically. Users don't have to involve in this process.

```python

a=7
print(type(a)) #python automatically converts a to int

b=3.0
print(type(b)) #Python automatically converts b to float

c=a+b #Python automatically converts c to float
      #as it is a float addition

d=a*b #Python automatically converts d to float
      # as it is a float multiplication

print(type(c))
print(type(d))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Videos\photography\nazia majadi\Data science Python code> pyt
<class 'int'>
<class 'float'>
<class 'float'>
<class 'float'>
PS C:\Users\DELL\Videos\photography\nazia majadi\Data science Python code>
```

2.Explicit Type conversion

Python needs user involvement to convert the variable data type into the required data type.

i)→Python Convert Int to Float

Converting Int to Float in Python with the **float()** function.

```python
a=5  #int variable

n=float(a) #typecast to float
print(n)
print(type(n))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data science
DELL\Videos\photography\nazia majadi\Data science Python code
5.0
<class 'float'>
```

ii)→Python Convert Float to Int

Converting Float to int datatype in Python with **int()** function.

```python
a=5.9  #int variable

n=int(a) #typecast to int
print(n)
print(type(n))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data scien
DELL\Videos\photography\nazia majadi\Data science Python co
5
<class 'int'>
```

iii)→Python Convert int to String

Converting int to String datatype in Python with **str()** function.

```python
a=5  #int variable

n=str(a) #typecast to str
print(n)
print(type(n))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data science Pythor
DELL\Videos\photography\nazia majadi\Data science Python code\commer
5
<class 'str'>
```

iv)→Python Convert String to float

casting string data type into float data type with **float()** function.

```python
a="6.8"  #string variable

n=float(a) #typecast to float
print(n)
print(type(n))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data science Pyt
DELL\Videos\photography\nazia majadi\Data science Python code\co
6.8
<class 'float'>
```

v)→Python Convert string to int

Converting string to int datatype in Python with **int()** function. If the given string is not number, then it will throw an error.

```python
a="5" # number
b="x" #is not a number

n=int(a) #typecast to int
print(n)
print(type(n))

n=int(b)  #if the given string is not a number,
print(b)  # then it will throw an error
print(type(b))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data science Python
DELL\Videos\photography\nazia majadi\Data science Python code\comment
5
<class 'int'>
Traceback (most recent call last):
  File "c:\Users\DELL\Videos\photography\nazia majadi\Data science Py
ne 8, in <module>
    n=int(b)  #if the given string is not a number,
ValueError: invalid literal for int() with base 10: 'x'
```

vi) → Addition of string and integer Using Explicit Conversion

```python
a=5 # number
b="x" #is not a number

n=a+b #typecast to int
print(n)
print(type(n)) #Type error
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data science P
DELL\Videos\photography\nazia majadi\Data science Python code\c
Traceback (most recent call last):
  File "c:\Users\DELL\Videos\photography\nazia majadi\Data scie
ne 4, in <module>
    n=a+b #typecast to int
TypeError: unsupported operand type(s) for +: 'int' and 'str'
PS C:\Users\DELL\Videos\photography\nazia majadi\Data science P
```

Python String Concatenation

i)→Merge variable `a` with variable `b` into variable `c`:

```
# method 1

a="Hello"
b=" world" #space
c=a+b
print(c) #two string variable concatenation
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data s
DELL\Videos\photography\nazia majadi\Data science Pytho
Hello world
PS C:\Users\DELL\Videos\photography\nazia majadi\Data s
```

ii)→To add a space between them, add a `" "`:

```
# method 2

a="Hello"
b="world" # without creating space inside
c=a +" "+ b
print(c) #two string variable concatenation
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data sci
DELL\Videos\photography\nazia majadi\Data science Python
Hello world
PS C:\Users\DELL\Videos\photography\nazia majadi\Data sci
```

Python - Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python keywords.

i)→Legal variable names:

```
#Legal variable names:

myvar = "John"
my_var="John"
_my_var="John"
myVar="John"
MYVAR="John"
myvar2="John"
```

ii)→Illegal variable names:

```
#illegal variable names:

2myvar ="John" #Error(2-number)
my-var="John"  #Error(-hypen)
my var="John"  #Error(  space)
```

###NOTE: Remember that variable names are case-sensitive

Multi Words Variable Names

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

i)→**Camel Case**

Each word, except the first, starts with a capital letter:

ii)→Pascal Case

Each word starts with a capital letter:

iii)→**Snake Case**

Each word is separated by an underscore character:

```
myVariableName = "John" #Camel Case

MYVariableName = "John" #Pascal Case

my_variable_name ="John" #Snake Case
```

Multiple variable assignment to the same string value in Python

i)→Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

```
x,y,z="orange","Banana","Cherry" #Many values to Multiple variables

print(x) #output:orange
print(y) #output:Banana
print(z) #output:Cherry
```

ii)→One Value to Multiple Variables

you can assign the *same* value to multiple variables in one line:

```
x = y = z="orange"

print(y) #output: orange
print(z) #output: orange
```

iii)→Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

```
fruits=["apple","banana","cherry"]
x,y,z=fruits #Unpack a list

print(x) #output: apple
print(y) #output: banana
print(z) #output: Cherry
```

Python Numbers

There are three numeric types in Python:

- `int`
- `float`
- `complex`

```
x=1    #int
y=2.8 #float
z=2j #complex

print(type(x))
print(type(y))
print(type(z))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data
DELL\AppData\Local\Temp\5f406235-c55c-472c-a25a-3dd0ce
omments.py"
<class 'int'>
<class 'float'>
<class 'complex'>
PS C:\Users\DELL\Videos\photography\nazia majadi\Data
```

i)→Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

```
x=1
y=3684884997868677888 #long integer value
z=-4757493            # negative value
print(x,y,z)
print(type(x))
print(type(y))
print(type(z))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data science
DELL\AppData\Local\Temp\5f406235-c55c-472c-a25a-3dd0cea21897_6
omments.py"
1 3684884997868677888 -4757493
<class 'int'>
<class 'int'>
<class 'int'>
```

ii)→Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

```
x=1.10
y=1.0
z=-35.59
print(x,y,z)
print(type(x))
print(type(y))
print(type(z))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data scienc
DELL\AppData\Local\Temp\5f406235-c55c-472c-a25a-3dd0cea21897
omments.py"
1.1 1.0 -35.59
<class 'float'>
<class 'float'>
<class 'float'>
```

Note: Float can also be scientific numbers with an "e" to indicate the power of 10.

```
x=35e3
y=12E4
z= -87.7e100
print(x,y,z)
print(type(x))
print(type(y))
print(type(z))
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data s
DELL\AppData\Local\Temp\5f406235-c55c-472c-a25a-3dd0cea
omments.py"
35000.0 120000.0 -8.77e+101
<class 'float'>
<class 'float'>
<class 'float'>
```

iii)→Complex

Complex numbers are written with a "j" as the imaginary part:

```python
x= 3+5j
y= 5j
z= -5j
print(x,y,z)
print(type(x))
print(type(y))
print(type(z))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Videos\photography\nazia majadi\Data s
DELL\AppData\Local\Temp\5f406235-c55c-472c-a25a-3dd0cea
omments.py"
(3+5j) 5j (-0-5j)
<class 'complex'>
<class 'complex'>
<class 'complex'>
```

Type conversion

You can convert from one type to another with the `int()`, `float()`, and `complex()` methods:

Convert from one type to another:

```python
x=1       #int
y=2.8     #float
z=2j      #complex

a=float(x)
b=int(y)
c=complex(x)

print(a,b,c)
print(type(a))
print(type(b))
print(type(c))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Videos\photography\nazia majadi\Da
DELL\AppData\Local\Temp\5f406235-c55c-472c-a25a-3dc
omments.py"
1.0 2 (1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

Addition of two numbers in scientific notation

```
x=35e3
y=-12E3
print(x+y) #scientific number addition
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data sc
DELL\AppData\Local\Temp\5f406235-c55c-472c-a25a-3dd0cea2
omments.py"
23000.0
```

Addition of two numbers in complext number

```
x= 3+5j
y=5j
print(x+y) #addition of two complex number
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data scier
DELL\AppData\Local\Temp\5f406235-c55c-472c-a25a-3dd0cea2189
omments.py"
(3+10j)
```

Python If ... Else

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b

- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

i)→If statement:

if statement" is written by using the if keyword.

```
a=33
b=200
if b>a:
    print("b is greater than a")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data
DELL\Videos\photography\nazia majadi\Data science Pyt
b is greater than a
```

ii)→**Elif**

The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

```
a=33
b=33
if b>a:
    print("b is greater than a")
elif a==b:
    print("a and b are equal")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data scie
DELL\Videos\photography\nazia majadi\Data science Python c
a and b are equal
```

iii)→**Else**

The else keyword catches anything which isn't caught by the preceding conditions.

```
a=200
b=33
if b>a:
    print("b is greater than a")
elif a==b:
    print("a and b are equal")

else:
    print("a is greater than b") #output: a is greater than b
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    Co

PS C:\Users\DELL\Videos\photography\nazia majadi\Data science Python code>
DELL\Videos\photography\nazia majadi\Data science Python code\comments.py"
a is greater than b
```

iv)→ `else` without the `elif`:

```
a=200
b=33
if b>a:
    print("b is greater than a")

else:
    print("b is not greater than a") #output: b is not greater than a
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    Code

PS C:\Users\DELL\Videos\photography\nazia majadi\Data science Python code> python
DELL\Videos\photography\nazia majadi\Data science Python code\comments.py"
b is not greater than a
```

v)→**Short Hand If**

If you have only one statement to execute, you can put it on the same line as the if statement.

```
a=200
b=33

if a>b: print("a is greater than b")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data
DELL\Videos\photography\nazia majadi\Data science Pyth
a is greater than b
```

vi)→Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

```
a=2
b=330

print("A") if a>b else print("B")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data s
DELL\Videos\photography\nazia majadi\Data science Pytho
B
```

```
a=2
b=330

print("A" if a>b else "B")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data
DELL\Videos\photography\nazia majadi\Data science Pyt
B
```

vii)→Multiple if statement

```python
a=330
b=200

if a>b:
    print("a is greater than b")
if a>=b:
    print("a is greater or equal than y")
if a==b:
    print("Equal")
else:
    print("Error")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Videos\photography\nazia majadi\Data science Pyth
DELL\Videos\photography\nazia majadi\Data science Python code\comm
a is greater than b
a is greater or equal than y
Error
```

viii)→Using pass in Conditional Statements

```python
x=10
if x<5:
    pass  #placeholder for future logic
else:
    print("x is 5 or less")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Videos\photography\nazia majadi\Data science
DELL\Videos\photography\nazia majadi\Data science Python code
x is 5 or less
```

ix)→Nested If

You can have `if` statements inside `if` statements, this is called *nested* `if` statements.

```python
x=41

if x>10:
    print("Above ten")
    if x>20:
        print("and also above 20!")
    else:
        print("but not above 20.")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data sci
DELL\Videos\photography\nazia majadi\Data science Python
Above ten
and also above 20!
```

Python operator

i)→And

The and keyword is a logical operator, and is used to combine conditional statements:

```python
a=200
b=33
c=500
if a>b and c>a:
    print("Both conditions are True")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data s
DELL\Videos\photography\nazia majadi\Data science Pytho
Both conditions are True
```

ii)→Or

The `or` keyword is a logical operator, and is used to combine conditional statements:

```
a=200
b=33
c=500
if a>b or a>c:
    print("At least one of the conditions is true")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data science Py
DELL\Videos\photography\nazia majadi\Data science Python code\co
At least one of the conditions is true
```

iii) → **Not**

The `not` keyword is a logical operator, and is used to reverse the result of the conditional statement:

```
a=33
b=200
if not a>b:
    print("a is not greater than b")
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Videos\photography\nazia majadi\Data sci
DELL\Videos\photography\nazia majadi\Data science Python
a is not greater than b
```