

Week 03

Python Programming Fundamentals

Conditions and Branching

`a == 6`

`i > 5`

`i >= 5`

`a != 6`

`"Ac" == "MJ" : False`

`if age > 18 :`

`print("you can enter")`

`print("move on")`

`if age > 18 :`

`print("you can enter")`

`else:`

`print("go see to next")`

`print("move on")`

```
if age > 18 :  
    print("you can enter")
```

```
elif (age == 18):  
    print("go see Pink Floyd")
```

```
else:  
    print("go see to next")  
print("move on")
```

Logic Operators

not (True) : False
not (False) : True.

True or True : True
True or False : True.
False or False : False

True and True : True
True and False : False
False and False : False.

```
if (a > 20) and (b > 10)  
    print(~~~)  
print(~~~)
```

Loops

range(3) : [0, 1, 2]

range(10, 15) : [10, 11, 12, 13, 14]

For loop

Squares = ["red", "yellow", "green", "purple", "blue"]

ex: for i in range(0, 5) :

Squares[i] = "white"

This replace values in Squares to white.

ex2 for s in Squares :

s


This prints all colors in Squares ?

ex3 for x in ["A", "B", "C"] :
print(x+A)

Output AA
BA
CA

Squares = ["red", "yellow", "green"]

for i, Square in enumerate(Squares):



Square
i

gives

"red"
0
"yellow"
1
"green"
2

for i, x in enumerate(['A', 'B', 'C']):
 print(i, x)

gives

0 A
1 B
2 C

While loop

Squares = ['orange', 'orange', 'purple', 'orange', 'blue']

New_squares = [] ← empty list

i = 0

While (Squares[i] == 'orange') :

New_squares.append(Squares[i])

i = i + 1

Functions

input \rightarrow \rightarrow output

len	\rightarrow	gives the length	len(x)
sum	\rightarrow	add values	sum(n)
sorted	\rightarrow	Sort List	sorted(n)
sort	\rightarrow	" "	x.sort()

Making functions.

```
def add1(a):  
    b = a + 1  
    return b
```

```
add1(5) : 6  
add1(10) : 11
```

```
def add1(a):  
    """  
    add 1 to a  
    """  
    b = a + 1  
    return b
```

} documentation.

```
help(add1) : add 1 to a
```

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

```
    c = a*b
```

```
    return c
```

```
Mult(2,3): 6
```

```
Mult(2,MJ): MJMJ
```

```
def MJ():
```

```
    print('Michael Jackson')
```

```
MJ: Michael Jackson.
```

```
def NoWork():
```

```
    pass
```

← nothing happens

```
print(NoWork()):
```

functions do more than one task

```
def add1(a):  
    b = a + 1  
    print(a, "plus 1 equals", b)  
    return b
```

Collecting arguments

```
def ArtistNames(*names):  
    for name in names:  
        print(name)
```

why?

reason for * we can call this fn

ArtistNames("Michael Jackson", "Dulithen")
or ArtistNames("Michael Jackson", "Dulithen", "Madhuri")

Scope : part of the program where that variables are accessible.

Global Scope

```
def AddDC(m):  
    x = x + "DC"  
    print(m)  
    return(m).
```

Global variable

→ x = "Ae"

z = AddD(m)

Global Scope

```
def Thriller():  
    Date = 1982  
    return Date
```

local

```
Thriller()  
Date
```

Global Scope

```
def Thriller():  
    Date = 1982  
    return (Date)
```

```
Date = 2017  
print (Thriller())  
print (Date)
```

Same variable name
Global and Local
No conflict

Output :
1982
2017

Global Scope

```
def ACD(y):  
    print (Rating)  
    return (Rating + y)
```

```
Rating = 9  
Z = ACD(1)  
print (Rating)
```

No local value for
the variable



python search it
globally, and use
global value

Output :

9
10
9

Global Scope

```
def PinkFloyd():
```

```
    global ClaimedSales
```

```
    ClaimedSales = 45
```

```
    return ClaimedSales
```

```
PinkFloyd()
```

```
print(ClaimedSales)
```

make it
Global

Exception Handling

```
a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
else:
    print("success a=", a)
finally:
    print("Processing Complete")
```

Please enter a number to divide a dulitha
You did not provide a number
Processing Complete

Objects and Classes

python data types

- int
- float
- String
- List
- Dictionary
- Bool

Each is an Object

Every Object has following

- Type
- internal data representation
- methods

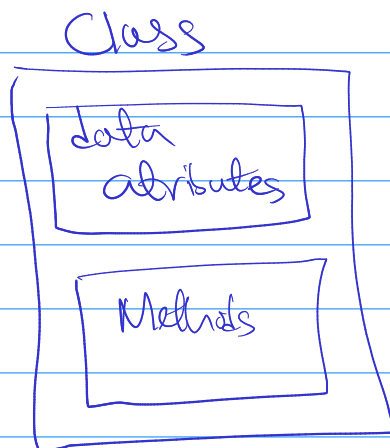
type() Command → gives the type.

Methods

functions
interact with object.

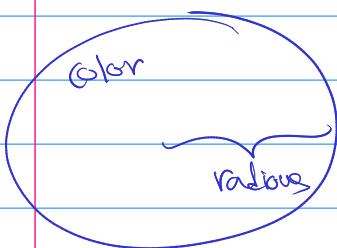
ex : sort / reverse
R = [10, 9, 6, 12]
R.sort()

Create a class



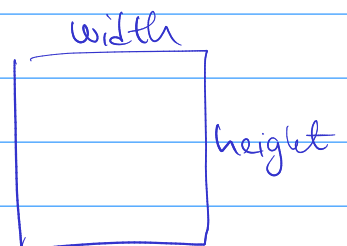
Then create objects on instances of that class

Class Circle.



Data attributes : radius, color

Class Rectangle



Data attributes } width
 height
 color

Build Circle Class

Define

```
# Create a class circle

class Circle(object):

    # Constructor
    def __init__(self, rad=3, col='blue'):
        self.radius = rad
        self.color = col

    # Method
    def add_radius(self, r):
        self.radius = self.radius + r
        return(self.radius)
```

class definition (syntax)

Name of the class

Class parents (syntax)

Syntax

Saying I am building new class

self parameter we don't worry about it

parameters

default

Data attributes

Creating an object from our class

name of the class

attributes

name of the object

Object constructor

```
Red10Circle = Circle(10, "red")
```

get attributes

```
Red10Circle.radius : 10
Red10Circle.color : "red"
```

Change attributes

```
Red10Circle.color = "blue"
```

Methods

Create a class Circle

```
class Circle(object):
```

```
-----
```

```
    # Constructor
```

```
    def __init__(self, rad=3, col='blue'):
```

```
        self.radius = rad
```

```
        self.color = col
```

```
-----
```

```
    # Method
```

```
    def add_radius(self, r):
```

```
        self.radius = self.radius + r
```

```
        return(self.radius)
```

} Method

Calling Methods

`red10Circle.add_radius(2)`

Change radius from 10 to 12


```
dir(RedCircle): "color"  
                "radius"  
                "add_radius"  
                "_class_"  
                "_m_" } don't worry
```