

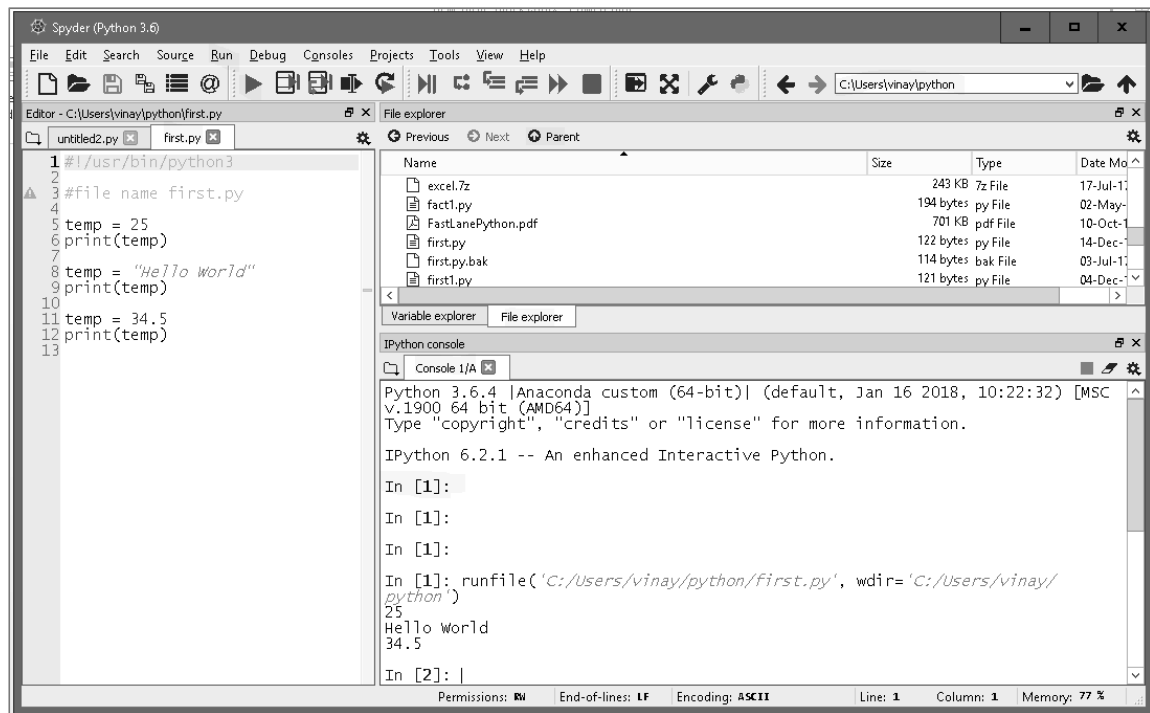
Python Programming Language

Features

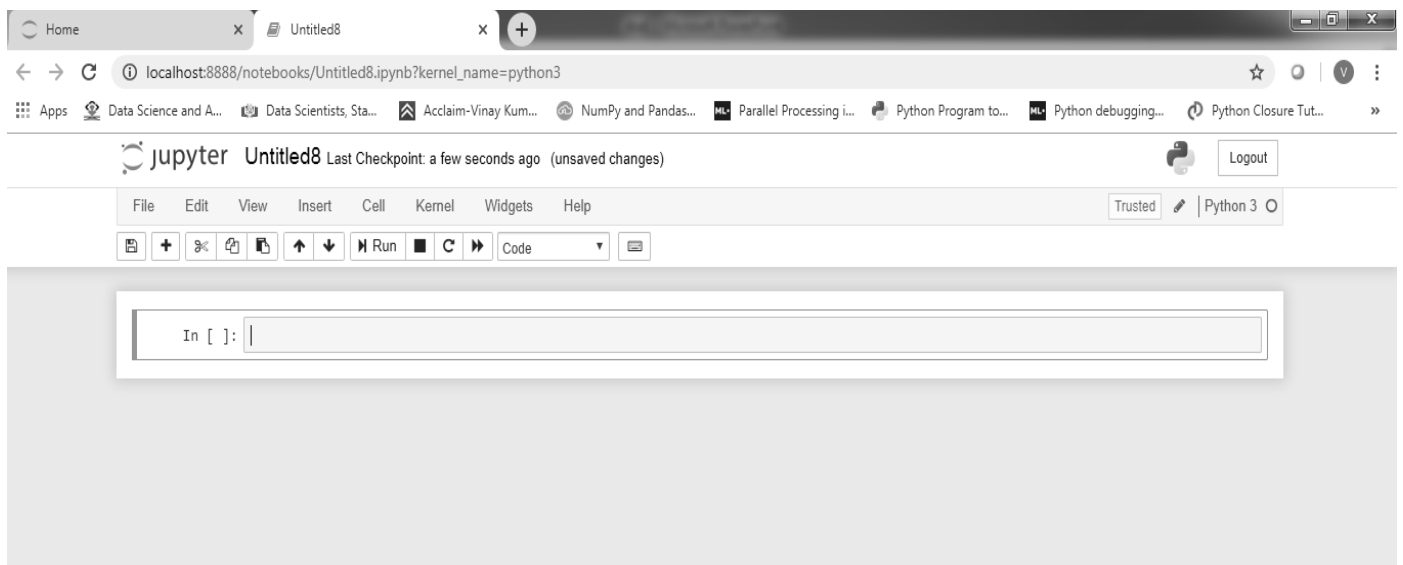
- **Simple Syntax:**
- **High-level Language**
- **Object Oriented**
- **Free and Open Source**

- **Extensive Libraries**
 - Data Science: Pandas, Numpy, Matplotlib
 - Web Frameworks: Django, Flask, Pyramid, Tornado.
 - Gaming: Pygame, piglet, Panda3D, PIL, GUI using Tk
 - Test Automation: unittest, pyTest
 - Database Access: pymysql, pymongo, openpyxl
 - Machine Learning: sklearn, Tensorflow..

Spyder IDE



Jupyter IDE



Statement Syntax...

```
>>> print("City Delhi") # Line Comment Starts
City Delhi
>>>
>>> #print("City Delhi")
>>>
>>> print("City # Delhi")
City # Delhi
>>>
```

```
>>> #Multiple Statements on Single Line
>>>
>>> temp = 25; temp1 = "Hello"
>>> print(temp); print(temp1)
25
Hello
>>>
```

Python Identifier

- A Python Identifier is a name to identify a Variable, function, class, module or any object.
- Case Sensitive. Starts with Alphabet. Should not start with Numbers.
- Should not have a Special Character like \$, @, #. underscores_ can be used.
- Python Language Keywords cannot be used as Identifier Names.

```
>>> temp = 22
>>> Temp = 33
>>> print(temp)
22
>>> print(Temp)
33
>>> temp456num = 4455
>>> temp_23_string = "Hello World"
>>>
```

```
>>> temp$me = 66
SyntaxError: invalid syntax
>>>
>>> 75temp = 32
SyntaxError: invalid syntax
>>>
>>> def = 45
SyntaxError: invalid syntax
>>>
```

Reserved Words

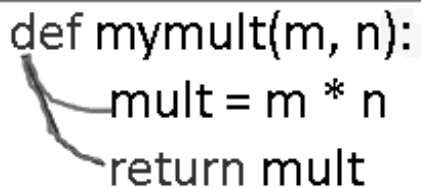
- Python has the following keywords or reserved words; they cannot be used as identifiers.

'False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',
'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield'

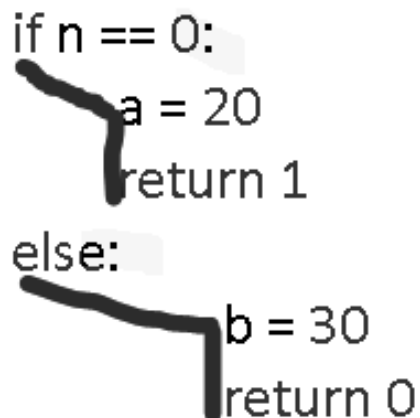
Indentation for Code Block.

- Indentation indicates Code-Blocks.

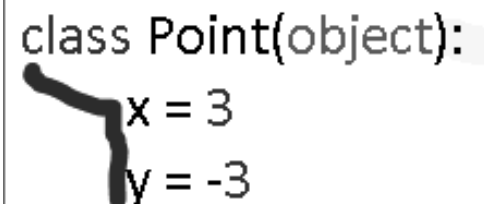
```
def mymult(m, n):  
    mult = m * n  
    return mult
```



```
if n == 0:  
    a = 20  
    return 1  
else:  
    b = 30  
    return 0
```



```
class Point(object):  
    x = 3  
    y = -3
```



Quotations

- Python accepts ('), (") and (""" or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
- The triple quotes are used to span the string across multiple lines.

```
word = 'my_word'
print(word)

sentence = "This is a sentence"
print(sentence)

paragraph = """ This is a paragraph made up of
multiple lines and sentences"""
print(paragraph)
```

Block Comments

- ```
name1 = "Hello"; print(name1)

"""
This is a block comment line1.
This is a block comment line2.
This is a block comment line3.
"""

temp = 22;
if (temp == 22):
 """
 Block comment 4
 Block comment 5
 """
 print(temp)
```

# Standard Data Types

- Python has five standard data types
  - Numbers
  - String
  - List
  - Tuple
  - Dictionary

## Assignment of Value

- No Separate Declaration Phase.
- Memory Allocated as soon as Definition.

```
myint = 23
myname = "Delhi"
myfloat = 45.678

print(myint)
print(myname)
print(myfloat)

a = b = c = 89
x,y,z = 34,"Hello",98.765
print(y)
```

## Type of a Identifier

- `type(..)` function is used to find out the type of the **object**.

```
a = b = c = 89
x,y,z = 34,"Hello",98.765
print(y)
```

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

```
Hello
<class 'int'>
<class 'str'>
<class 'float'>
```

## Numbers Types

- Number objects are created when you assign a numerical value to an identifier.

```
>>>
>>> num = 23456
>>> type(num)
<class 'int'>
>>> numf = 234.567
>>> type(numf)
<class 'float'>
>>> numc = 3.14j
>>> type(numc)
<class 'complex'>
>>>
```

## Strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Not-Mutable.

```
>>> mystr1 = 'HelloWorld'
>>> mystr2 = "Thank You"
>>> type(mystr1)
<class 'str'>
>>> type(mystr2)
<class 'str'>
>>> print(mystr1,mystr2)
HelloWorld Thank You
```

## List

- Sequence of Items, [... , ... , ....]
- Different data types, Individual Access, Mutable.

```
>>> mylist = [234,5.67,"Hello",567]
>>> type(mylist)
<class 'list'>
>>> print(mylist)
[234, 5.67, 'Hello', 567]
>>> print(mylist[2])
Hello
>>> type(mylist[2])
<class 'str'>
```



# Tuple

- Sequence of Items, (... , ... , ....)
- Different data types, Individual Read Access, Not-Mutable.

```
>>>
>>> mytuple = (234, 45.6, "Hello")
>>> type(mytuple)
<class 'tuple'>
>>> print(mytuple)
(234, 45.6, 'Hello')
```

# Dictionary

- Dictionary datatypes , defines 1:1 relationships between keys and values.
- Each key is separated from its value by a colon (key:value).

```
>>> studenthash = {"Name":"Python", "ID":1234, "City":"Delhi", 91:"India"}
>>> type(studenthash)
<class 'dict'>
>>> print(studenthash)
{'ID': 1234, 'City': 'Delhi', 91: 'India', 'Name': 'Python'}
>>> print(studenthash["ID"])
1234
>>> print(studenthash[91])
India
```

## Typical print statement

```
numd = 22
str1 = "Windows10"
```

```
print(numd, str1)
```

```
print("We have %d number of Licenses of %s " %(numd, str1))
```

```
print("We have {} number of Licenses of {}".format(numd, str1))
```

## Escape Sequences

- Escape sequences like `\r`, `\n`, `\t` allow users to communicate with a display device or printer by sending non-graphical control characters to specify actions like newline, tabs, carriage returns, etc.

|                                      |                                |
|--------------------------------------|--------------------------------|
| <code>print("Hello World")</code>    | <code>## Regular String</code> |
| <code>print("Hello \t World")</code> | <code># Tab</code>             |
| <code>print("Hello \n World")</code> | <code># New Line</code>        |
| <code>print("Hello \r World")</code> | <code># Carriage Return</code> |
| <code>print("Hello \v World")</code> | <code># Vertical Tab</code>    |
| <code>print("Hello \" World")</code> | <code># Quotes</code>          |
| <code>print("Hello \bWorld")</code>  | <code># Backspace</code>       |
| <code>print("c: \\tiger")</code>     | <code># Backslash</code>       |

```
Hello World
Hello World
Hello
 World
 World
Hello
 World
Hello " World
Helloworld
c:\tiger
```

## Keyboard Input. 3.x version

```
.

>>> userInput = input("Enter your Name ")
Enter your Name Python
>>> type(userInput)
<class 'str'>
>>> print(userInput)
Python
```

```
>>> userInput = input("Enter your Age ")
Enter your Age 25
>>> type(userInput)
<class 'str'>
>>> age = int(userInput)
>>> type(age)
<class 'int'>
>>> print(age)
25
```

```
>>> userInput = int(input("Enter your Age "))
Enter your Age 25
>>> type(userInput)
<class 'int'>
```

## Operators

#Arithmetic  
#Operators

```
x = 15; y = 2

a = x + y
s = x - y
m = x * y
df = x / y
dq = x // y
dr = x % y
p = x ** y
```

#Relational  
#Operators

```
x = 15; y = 2

br = x == y

if y == z:
if x != y:
if x > y:
if x < y:
if x >= y:
if x <= y:
```

#Assignment  
#Operators

```
y = 2

x = y
x = x + y; x += y
x = x - y; x -= y
x = x * y; x *= y
```

#Logical  
#Operators

```
x = 5

if x > 0 and x > 4:
if x > 0 or x > 4:
if not(x):
if x:
```

# Operators

#Bitwise  
#Operators

```
x = 5; y = 3
```

```
a = x & y
```

```
a = x | y
```

```
a = x ^ y
```

```
a = ~x
```

```
a = x << 2
```

```
a = x >> 3
```

```
print("{0:b}".format(a))
```

#Membership  
#Operators

```
br = "good" in "this is a good book"
if "good" in "this is a excellent book"
```

```
block_list = ["www.crik.com", "www.movis.com"]
if "www.crik.com" in block_list:
if 'www.itoday.com' not in block_list:
```

#Identity  
#Operator

```
a = 23; b = 23
br = a is b
br = a is not b
```

## Overall Operator Precedence

•

| Operator                 | Description                                 |
|--------------------------|---------------------------------------------|
| **                       | Exponentiation (raise to the power)         |
| ~ + -                    | Complement, unary plus and minus            |
| * / % //                 | Multiply, divide, modulo and floor division |
| + -                      | Addition and subtraction                    |
| >> <<                    | Right and left bitwise shift                |
| &                        | Bitwise 'AND'                               |
| ^                        | Bitwise exclusive 'OR' and regular 'OR'     |
| <= < > >=                | Comparison operators                        |
| <> == !=                 | Equality operators                          |
| = %= /= //= -= += *= **= | Assignment operators                        |
| is is not                | Identity operators                          |
| in not in                | Membership operators                        |
| not or and               | Logical operators                           |

# Function Syntax

- A Function is a named sequence of statements that performs a desired operation.
  - This sequence is specified in a function definition.
- Functions are loaded in memory on definition and Interpreted when called.

```
def myFunction(a,b,c):
 s = a + b + c
 print("Args are %d, %d, %d " %(a,b,c))
 print("Sum of Args is ", s)
 return s

x = myFunction(2,4,6)
print(x)
```

# Slicing

- To access substrings, use the **square brackets [ ]** for Slicing along with the index or indices to obtain your substring.

- mystring[N:M:STEP]

- returns part of the string from,  
Nth character to the Mth character, excluding  
Mth Character, with step size of STEP..

| Indexing from Start of String -----> |     |     |    |    |    |    |    |    |    |    |    |
|--------------------------------------|-----|-----|----|----|----|----|----|----|----|----|----|
| 0                                    | 1   | 2   | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 |
| A                                    | B   | C   | D  | E  | F  | G  | H  | I  | J  | K  | L  |
| -12                                  | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |
| <-----Indexing from End of String    |     |     |    |    |    |    |    |    |    |    |    |

```
mystr = "ABCDEFGHIIKL"
print(mystr[-1])
print(mystr[5])
print(mystr[-5])
print(mystr[2:8])
print(mystr[8:2])
print(mystr[-8:-2])
```

```
print(mystr[-2:-8])
print(mystr[1:-8])
print(mystr[-8:1:-1])
print(mystr[2:10:2])
print(mystr[1:])
print(mystr[:8])
print(mystr[:])
print(mystr[::-1])
```

## Concatenation

```
s1 = "Python "
s2 = "Programming"
s3 = s1 + s2
s4 = s1 * 3
```

```
s1 = "Python "
leng = len(s1)
```

## string.find

```
>>> mystr = "Statement"
>>>
>>> retval = mystr.find('a')
>>> print(retval)
2
>>>
>>> print(mystr.find('t'))
1
>>>
>>> print(mystr.find('t',2))
3
```

```
>>> print(mystr.find('ate'))
2
>>>
>>> print(mystr.find('t',4,7))
-1
>>>
>>> print(mystr.rfind('t'))
8
```

## String strip functions

- `.strip()` removes whitespace chars on both ends
- `.lstrip()` removes whitespace chars on left end.
- `.rstrip()` removes whitespace chars on right end.

```
mystr2 = " ::cap tap sap map lap:: "
print(mystr2); print(len(mystr2))

newstr = mystr2.strip()
print(newstr); print(len(newstr))
print(newstr.strip(':'))

print(mystr2.lstrip())
print(mystr2.rstrip())
```

## Split and Join

- `.split(',')` splits a string on ',' and the return a list of words. Whitespace by default.
- `.split(',',2)` splits a string on ',' only two times and the return a list of words.
- `'x'.join(y)` joins every element in the list y separated by 'x'

```
mystr3 = "ban,can,fan#lan,man#pan,ran,tan,van zan"
mwords = mystr3.split()
print(mwords); print(len(mwords))

mwords = mystr3.split(',')
print(mwords); print(len(mwords))
```

```
mwords = mystr3.split(',',2)
print(mwords); print(len(mwords))

mwords = mystr3.split(',',3)
print(mwords); print(len(mwords))

print(mystr3)
mwords = mystr3.split(',')
print(mwords)
newword = ':'.join(mwords)
print(newword)
```

## Lists: Definition

- A list is an **ordered set of values**, where each value is identified by an **index**.
  - The list **Elements** are enclosed in Square Brackets. [ ]

```
cities= ["Delhi", "Bangalore", "Mumbai"]
num = [17, 123,243]
emptylist = []
mlist = ["Hello", 123, (20,"World"), [1,"Jan",(2017,2018)]]
print(mlist, type(mlist))
print(mlist[2], type(mlist[2]))
print(mlist[-1], type(mlist[-1]))
print(mlist[3][2], type(mlist[3][2]))
print(mlist[-1][2][1], type(mlist[-1][2][1]))
```

## Lists are Mutable

- Unlike strings, lists are mutable, which means we can change their elements.
- Using the bracket operator on the left side of an assignment, we can update one of the elements:
  - >>> fruit = ["banana", "apple", "quince"]
  - >>> fruit[0] = "pear"
  - >>> fruit[-1] = "orange"
  - >>> print(fruit)
  - >>> a = [10,20,30]
  - >>> del a[1]                      # del function removes element from a list
  - >>> list1 = ['a', 'b', 'c', 'd', 'e', 'f']
  - >>> del list1[1:5]



## Methods of Lists

|                                               |                                                              |
|-----------------------------------------------|--------------------------------------------------------------|
| <code>mylist1 = [2,4,6,3,5,7,1];</code>       | <code>print(mylist1)</code>                                  |
| <code>mylist1.append(21);</code>              | <code>print("Append an element into", mylist1)</code>        |
| <code>mylist1.extend([5,10,15]);</code>       | <code>print("extend", mylist1)</code>                        |
| <code>mylist1.insert(3,(77,88,99));</code>    | <code>print("insert into an index", mylist1)</code>          |
| <code>mylist1.pop(-5); mylist1.pop();</code>  | <code>print("pop", mylist1)</code>                           |
| <code>mylist1.remove(21);</code>              | <code>print("remove value", mylist1)</code>                  |
| <code>mylist1.reverse();</code>               | <code>print("reverse", mylist1)</code>                       |
| <code>print(mylist1.index(5),end=',');</code> | <code>print(mylist1.index(5,2,7),end=',')</code>             |
| <code>print(mylist1.count(5));</code>         | <code>print(mylist1.count(2))</code>                         |
| <code>mylist2 = mylist1.copy();</code>        | <code>print(mylist2); mylist2[2] = 22; print(mylist1)</code> |
| <code>mylist1 = [2,4,6,3,5,7,1];</code>       | <code>print(mylist1)</code>                                  |
| <code>mylist1.sort();</code>                  | <code>print(mylist1)</code>                                  |
| <code>mylist1.sort(reverse=True);</code>      | <code>print(mylist1)</code>                                  |
| <code>mylist1.clear();</code>                 | <code>print(mylist1)</code>                                  |

## Tuples

- Immutable

```
mytuple = (1,2,3,'a','b','c',[10,11,12], (22,33,44))
print(mytuple); print(type(mytuple))
print(mytuple[2], mytuple[6], mytuple[7][1])

mytuple1 = (123,); print(mytuple1)
```

## Tuple Operations

```
>>> mt1 = (2,4,6)
>>> mt2 = (1,3,5)
>>> mt3 = mt1+mt2
>>> print(mt3)
(2, 4, 6, 1, 3, 5)
>>>
>>> mt4 = mt1*3
>>> print(mt4)
(2, 4, 6, 2, 4, 6, 2, 4, 6)
```

```
>>> mytuple = (1,2,3,4,5)
>>> mytuple[-1]
5
>>> mytuple[:]
(1, 2, 3, 4, 5)
>>> mytuple[2:5]
(3, 4, 5)
>>> mytuple[:5]
(1, 2, 3, 4, 5)
>>> mytuple[2:]
(3, 4, 5)
```

```
>>> mt1 = (2,4,6)
>>> mt2 = (1,3,5)
>>> mt1,mt2 = mt2,mt1
>>> print(mt1,mt2)
(1, 3, 5) (2, 4, 6)
```

```
>>> mt1 = (1,3,5,7,9)
>>> 3 in mt1
True
>>> 4 in mt1
False
```

## Tuple Operations

```
>>> mt1
(1, 3, 5, 7, 9)
>>> mlist1 = list(mt1)
>>> mlist1
[1, 3, 5, 7, 9]
>>>
>>> mt2 = tuple(mlist1)
>>> mt2
(1, 3, 5, 7, 9)
```

```
>>> mt1 = (3,5,7,3,1,19,3,11)
>>> len(mt1)
8
>>> max(mt1)
19
>>> min(mt1)
1
>>> mt1.count(3)
3
>>> mt1.index(3)
0
```

## Dictionary: Definition

- defines 1:1 relationships between keys and values.
- Each key is separated from its value by a colon {key:value}.
- Keys must be of an **immutable** data type
- Keys are **unique** within a dictionary while values may not be.
- Access into Dictionary is only thru Keys.

```
myd1 = {"Name": "James", "Age" : 5, "Height" : 3, 91: "India"}
print(myd1, myd1["Age"])
print(myd1[91])
```

```
myd2 = {(70,80,90):"distinction", (60):"first", (40,50):"pass", (10,20,30):"fail"}
print(myd2)
```

## Dictionary Update

```
>>> myd1 = {"Name": "James", "Age": 5, "Height" : 3, 91: "India"}
>>> print(myd1)
{'Name': 'James', 'Age': 5, 'Height': 3, 91: 'India'}
>>> myd1["School"] = "KVS"
>>> print(myd1)
{'Name': 'James', 'Age': 5, 'Height': 3, 91: 'India', 'School': 'KVS'}
>>> myd1["Height"] = 4
>>> print(myd1)
{'Name': 'James', 'Age': 5, 'Height': 4, 91: 'India', 'School': 'KVS'}
>>> del myd1["Age"]
>>> print(myd1)
{'Name': 'James', 'Height': 4, 91: 'India', 'School': 'KVS'}
>>> myd1.clear()
>>> print(myd1)
{}
```

# Dictionary Methods

```
>>> myd1 = {"Name": "James", "Age": 5, "Height" : 3, 91: "India"}
>>> len(myd1)
4
>>>
>>> myd1.keys()
dict_keys(['Name', 'Age', 'Height', 91])
>>> myd1.values()
dict_values(['James', 5, 3, 'India'])
>>>
>>> myd1.items()
dict_items([('Name', 'James'), ('Age', 5), ('Height', 3), (91, 'India')])
>>>
>>> mytuple1 = ("Game", "Player", "Age")
>>> myd2 = dict.fromkeys(mytuple1)
>>> myd2
{'Game': None, 'Player': None, 'Age': None}
>>> myd2 = dict.fromkeys(mytuple1, 22)
>>> myd2
{'Game': 22, 'Player': 22, 'Age': 22}
```

## if...elif...else construct

```
var1 = 45
if var1 >= 40:
 print("var1 Pass")

var2 = 35
if var2 >= 40:
 print("var2 Pass")
else:
 print("var2 Fail")
```

```
var3 = 75
if var3 >= 70:
 print("var3 distinction")
elif var3 >= 60:
 print("var3 first class")
elif var3 >= 40:
 print("var3 pass class")
else:
 print("var3 fail")
```

```
var4 = 65
if var4 >= 40:
 if var4 >= 70:
 print("var4 distinction")
 elif var4 >= 60:
 print("var4 first class")
 else:
 print("var4 pass class")
else:
 print("var4 fail")
```

# range

- We can generate a sequence of numbers using **range()** function.
- Three Signatures:
  - **range(END)** : generates numbers from 0 to END (excluding END), with step size of +1
  - **range(START, END)**: generates numbers from START to END, with step size of +1
  - **range(START, END, STEP)**: generates numbers from START to END, with step size of STEP.
- END value is excluded from generation

```
>>> r = range(7)
>>> type(r)
<class 'range'>
>>> x = list(r)
>>> print(x)
[0, 1, 2, 3, 4, 5, 6]
```

```
>>> print(list(range(5)))
[0, 1, 2, 3, 4]
>>>
>>> print(list(range(5,15)))
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

```
>>> print(list(range(5,15,2)))
[5, 7, 9, 11, 13]
>>>
>>> print(list(range(5,14,2)))
[5, 7, 9, 11, 13]
```

# Loops / Iterations

```
1
lcount = 10
while(lcount > 0):
 print(lcount)
 lcount = lcount - 1
```

```
2
for letter in "Python":
 print(letter)
```

```
3
stocks = ["infy", "tcs", "ongc"]
for ustock in stocks:
 print(ustock, end=':')
else:
 print("Traversed all Stocks")
```

```
4
for item in (1, 2, 'a', 'x', [3,6,8,'k']):
 print(item)
```

```
5
for num in range(10):
 print(num)
```

```
6
stocks = ("infy", "tcs", "ongc")
for indx in range(len(stocks)):
 print(stocks[indx], end=' ')
```

```
7
d1={'k1':'v44','k2':'v55','k3':'v66'}
print(d1)
for k in d1.keys():
 print(k)

for v in d1.values():
 print(v)

for k,v in d1.items():
 print(k, v)
```

# Break / Continue statement

- **break :**

- Loop Termination.

- **continue :**

- Abort Current Iteration
- Force New Iteration Restart.

```
for letter in 'Python':
 if letter == 'h':
 break
 print(letter, end="")

print()

var = 9
while var > 0:
 print(var, end=' ')
 var = var - 1
 if var == 5:
 break

print()
```

```
for letter in 'Python':
 if letter == 'h':
 continue
 print(letter, end="")

print()

var = 10
while var > 0:
 var = var - 1
 if var == 5:
 continue
 print(var, end=" ")
```

## Required, Keyword, Default arguments

- **Required arguments:** arguments passed in correct positional order.
  - Number of arguments in the function call should match with the function definition.
- **Keyword arguments:** caller identifies the arguments by parameter name.
  - Arguments can be out of Order, but need to be aware of the Parameter name
- **Default argument:** Argument that assumes a default value if a value is not provided in the function call for that argument.

```
def printinfo2(name, num=5):
 print (name,num)
 return;

printinfo2("James",15)
printinfo2(num=12, name="John")
printinfo2("Jack")
```

```
James 15
John 12
Jack 5
>>>
```

## The Anonymous Functions `lambda`

- Used to create Anonymous function objects.

```
addsum = lambda x, y : x + y
ms = addsum(2,5)
print("Total", ms)
print("Total:", addsum(10,20))
```

- Lambda effectively used in map, filter and reduce methods.

## Variable-length arguments

- processing a function for more arguments than specified.
  - These args are called variable-length arguments and are not named in the function definition.
  - An asterisk (\*) is placed before the tuple variable name, that holds the values of all non-keyword variable arguments.
  - This tuple remains empty if no additional arguments are specified during the function call.

```
def printinfo3(arg1,*vartuple):
 print (arg1),
 for var in vartuple:
 print(var, end=" ")
 return;

printinfo3(40)
print()
printinfo3(70, 60, 50)
```

```
40
70
60 50
>>>
```