Simple command: To excecute we can also use shift+enter

Python Introduction

Python is a cross-platform programming language, means, it runs on multiple platforms like Windows, Mac OS X, Linux, Unix. There are no type declarations of variables, parameters, functions, or methods in source code. This makes the code short and flexible, and you lose the compile-time type checking of the source code. Python tracks the types of all values at runtime.

In [ ]:

In [ ]:

In [1]:
```
print('Hello!')
```

```
Hello!
```

In [2]:
```
print("Hello World!")
```

```
Hello World!
```

```
You can use python as Calculator
```

In [6]:
```
3+4
```

Out[6]:
```
7
```

operators

In [7]:
```
print(2+3) #addition
print(5-2) #subtraction
print(5/2) #division
print(5*2) #multiplication
print(5//2) #modulus or quotient
print(5%2) #remainder
5**4
```

```
5
3
2.5
10
2
1
```

Out[7]:
```
625
```

Python Variables

A variable is a location in memory used to store some data (value). Multiple assignments are allowed in python.

In [26]:
```
a,b=10.5,20.5
print(a,b)
```

```
10.5 20.5
```

Native Data types in Python

Data types are the classification or categorization of data items. Python has the following standard or built-in data types:

Booleans are either True or False. **Numbers** can be integers (1 and 2), floats (1.1 and 1.2), fractions (1/2 and 2/3), or even complex numbers. **Strings** are sequences of Unicode characters, e.g. an html document. **Bytes and byte arrays**, e.g. a jpeg image file. **Lists** are ordered sequences of values. **Tuples** are ordered, immutable sequences of values. **Sets** are unordered bags of values. **Dictionaries** are unordered bags of key-value pairs.

Type *Markdown* and LaTeX: α^2

Numeric type

Python supports integers, floating point numbers and complex numbers. They are defined as int, float and complex class in Python.

Integer

Positive or negative whole numbers (without a fractional part)

In [3]:
```
a=-20
print(a)
```

```
-20
```

In [9]:
```
type(a)
```

Out[9]:
```
int
```

Float

Any real number with a floating point representation in which a fractional component is denoted by a decimal symbol or scientific notation

In [9]:
```
b=1.22e3
print(b)
type(b)
```

```
1220.0
```

Out[9]:
```
float
```

In [8]:
```
c=1.0002
print(c)
```

```
1.0002
```

In [8]:
```
type(b)
```

Out[8]:
```
float
```

Complex Number

A number with a real and imaginary component represented as x+yj.

In [6]:
```
c=2+2j
print(c)
```

```
(2+2j)
```

In [7]:
```
type(c)
```

Out[7]:
```
complex
```

Converting Integers To Floats And Vice-Versa

In [10]:
```
print(int(2.36))
print(float(26))
```

```
2
26.0
```

Binary, Octal, Hexadecimal

In [11]:
```
d=0b110
print(d)
```

```
6
```

In [13]:
```
e=0o14
print(e)
type(e)
```

```
12
```

Out[13]:
```
int
```

In [14]:
```
f=0xe
print(f)
```

```
14
```

Boolean

Data with one of two built-in values True or False. Notice that 'T' and 'F' are capital.

In [16]:
```
a=True
type(a)
```

Out[16]:
```
bool
```

In [29]:
```
5==5
```

Out[29]:
```
True
```

In [30]:
```
5==6
```

Out[30]:
```
False
```

boolean comparison operators:

x == y #x is equal to y

x != y #x is not equal to y

x > y #x is greater than y

x < y #x is less than y

x >= y #x is greater than or equal to y. Similarly, <=

x is y #x is the same as y

x is not y #x is not the same as y

```
Logical operators:
and
or
not
```

String

A string value is a collection of one or more characters

In [15]:
```
a="Hi"
print(a)
```

```
Hi
```

In [15]:
```
a='17'
type(a)
```

Out[15]:
```
str
```

In [12]:
```
b='How are you?'
print(b)
type(b)
print(b[-4])
```

```
How are you?
y
```

In [13]:
```
print(b[-4])
```

```
y
```

In [19]:
```
p="""This is a paragraph.
it can be written in python"""
print(p)
```

```
This is a paragraph.
it can be written in python
```

[] is a slice operator used to retrieve pieces of a string.

In [22]:
```
print(b[0])
print(b[4:7])
print(b[-1])
```

```
H
are
?
```

In [14]:
```
print(b*3)
```

```
How are you?How are you?How are you?
```

In [ ]:
```

```

In [26]:
```
print(a+b)
```

```
17How are you?
```

in operator returns True if a character exists in the given string

In [17]:
```
print('r' in b)
```

```
True
```

not in - operator returns true if a character does not exist in the given string

In [32]:
```
print('x' not in b)
```

```
True
```

In [36]:
```
print("Hello\tworld")
```

```
Hello	world
```

Methods for string

In [22]:
```
str="This is India"
str1="England"
print(str.upper())
print(str1.upper())
```

```
THIS IS INDIA
ENGLAND
```

In [19]:
```
print(len(str))
```

```
13
```

In [42]:
```
str.lower()
```

Out[42]:
```
'this is india'
```

In [43]:
```
str.capitalize()
```

Out[43]:
```
'This is india'
```

In [44]:
```
str.title()
```

Out[44]:
```
'This Is India'
```

In [23]:
```
print(str)
str.islower()
```

```
This is India
```

Out[23]:
```
False
```

In [46]:
```
str1="2000"
str1.isdigit()
```

Out[46]:
```
True
```

In [47]:
```
str.count('i') #counts the no of occurrences of 'i'
```

Out[47]:
```
3
```

In [48]:
```
str.find('is') #finds the first occurrence of a substring in the string
```

Out[48]:
```
2
```

In [ ]:
```

```

List

A list can also contain elements of different types It is mutable

In [2]:
```
ar=[1,2,33.7, "Mark", "Alice", True]
print(ar)
type(ar)
```

```
[1, 2, 33.7, 'Mark', 'Alice', True]
```

Out[2]:
```
list
```

In [54]:
```
ar[2]
```

Out[54]:
```
33.7
```

In [55]:
```
ar[-2]
```

Out[55]:
```
'Alice'
```

In [3]:
```
ar[3][2]
```

Out[3]:
```
'r'
```

In [56]:
```
ar[1]="Bob"  #mutable
print(ar)
```

```
[1, 'Bob', 33.7, 'Mark', 'Alice', True]
```

In [57]:
```
lst=[ar,20,30] #nested list
print(lst)
```

```
[[1, 'Bob', 33.7, 'Mark', 'Alice', True], 20, 30]
```

In [58]:
```
lst[0]
```

Out[58]:
```
[1, 'Bob', 33.7, 'Mark', 'Alice', True]
```

In [25]:
```
lst1=[[1,2,3],[10,20,30],["Harry","Bob"]]
lst1[2][0]
```

Out[25]:
```
'Harry'
```

List operations

In [26]:
```
a=[1,2,3]
b=[10,20,30]
c=a+b
print(c)
```

```
[1, 2, 3, 10, 20, 30]
```

In [63]:
```
a*2
```

Out[63]:
```
[1, 2, 3, 1, 2, 3]
```

In [64]:
```
c[1:3]
```

Out[64]:
```
[2, 3]
```

In [66]:
```
c[:4]
```

Out[66]:
```
[1, 2, 3, 10]
```

In [67]:
```
c[3:]
```

Out[67]:
```
[10, 20, 30]
```

In [68]:
```
c[:]
```

Out[68]:
```
[1, 2, 3, 10, 20, 30]
```

List methods

```
append() - Add an element to the end of the list
extend() - Add all elements of a list to the another list
insert() - Insert an item at the defined index
remove() - Removes an item from the list
pop() - Removes and returns an element at the given index
clear() - Removes all items from the list
index() - Returns the index of the first matched item
count() - Returns the count of number of items passed as an argument
sort() - Sort items in a list in ascending order
reverse() - Reverse the order of items in the list
copy() - Returns a shallow copy of the list
```

In [27]:
```
print(c)
```

```
[1, 2, 3, 10, 20, 30]
```

In [30]:
```
c.append("Bob")
print(c)
```

```
[1, 2, 3, 10, 20, 30, 'Bob']
```

In [31]:
```
c.pop(1)
print(c)
```

```
[1, 3, 10, 20, 30, 'Bob']
```

In [32]:
```
d=["Ali", 20]
c.append(d) #append adds a new element to the end of a list
print(c)
```

```
[1, 3, 10, 20, 30, 'Bob', ['Ali', 20]]
```

In [76]:
```
print(c[-1])
```

```
['Ali', 20]
```

In [77]:
```
c.pop(-1) #delete elements from a list
print(c)
```

```
[1, 2, 3, 10, 20, 30]
```

In [33]:
```
print(c)
c.extend(d) #extend takes a list as an argument and appends all of the elements
print(c)
```

```
[1, 3, 10, 20, 30, 'Bob', ['Ali', 20]]
[1, 3, 10, 20, 30, 'Bob', ['Ali', 20], 'Ali', 20]
```

In [79]:
```
t = ['d', 'c', 'e', 'b', 'a']
t.sort() #sort arranges the elements of the list from low to high
print(t)
```

```
['a', 'b', 'c', 'd', 'e']
```

Conversion of string to a list

In [82]:
```
str="Rugved"
lst=list(str)
print(lst)
```

```
['R', 'u', 'g', 'v', 'e', 'd']
```

Type *Markdown* and LaTeX: α^2

In [101]:
```
a="banana"
b="banana"
a is b
```

Out[101]:
```
True
```

In [34]:
```
x=[1,2,3] # both are different objects
y=[1,2,3]
x is y
```

Out[34]:
```
False
```

In [103]:
```
x=[1,2,3] # both variables are pointing same object
y=x
x is y
```

Out[103]:
```
True
```

Functions on List

In [104]:
```
num=[1,2,4,5,10,20]
print(len(num))
print(max(num))
print(min(num))
print(sum(num)) #works only on list containing numbers
```

```
6
20
1
42
```

*Arrays are special type of list where all elements are of same types.

Tuple-constructor

Tuple is a collection of items of any Python data type immutable

In [91]:
```
item=(1, "Jeff", "Computer", 75.50, True)
```

In [92]:
```
item[-2]
```

Out[92]:
```
75.5
```

In [93]:
```
item[1]=2
print(item)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-93-0d1a97e05173> in <module>
----> 1 item[1]=2
      2 print(item)

TypeError: 'tuple' object does not support item assignment
```

In [94]:
```
print(item)
```

```
(1, 'Jeff', 'Computer', 75.5, True)
```

In [105]:
```
t=('a')
type(t)
```

Out[105]:
```
str
```

In [106]:
```
t1=('a',) #To create a tuple with a single element, you have to include the final
type(t1)
```

Out[106]:
```
tuple
```

In [107]:
```
p=tuple() #creates an empty tuple
print(p)
```

```
()
```

In [35]:
```
t = tuple('hello')
print(t)
```

```
('h', 'e', 'l', 'l', 'o')
```

You can’t modify the elements of a tuple, but you can replace one tuple with another.

In [28]:
```
t =('A',) + t[1:]
print(t)
```

```
('A', 'e', 'l', 'l', 'o')
```

In [29]:
```
q=('B',)+t
print(q)
```

```
('B', 'A', 'e', 'l', 'l', 'o')
```

```
### Tuple operations
+ concatenation
* repeatation
[] slicing
in
not in
```

In [33]:
```
item=(1, "Jeff", "Computer", 75.50, True)
itm2=(1,2,"HR", 2.5)
i1=item+itm2
i2=itm2*2
i3=item[1:4]
print(i1)
print(i2)
print(i3)
```

```
(1, 'Jeff', 'Computer', 75.5, True, 1, 2, 'HR', 2.5)
(1, 2, 'HR', 2.5, 1, 2, 'HR', 2.5)
('Jeff', 'Computer', 75.5)
```

In [ ]:

In [98]:
```
1 in item
```

Out[98]:
```
True
```

In [99]:
```
a= 1 in item
print(a)
```

```
True
```

In [100]:
```
3 not in itm2
```

Out[100]:
```
True
```

Set Data type

A set is an unordered collection of data types. It has suitable methods to perform mathematical set operation

In [36]:
```
st1={1,2,2,3,4,4,2,5,1} #A set doesn't store duplicate objects.
print(st1)
type(st1)
```

```
{1, 2, 3, 4, 5}
```

Out[36]:
```
set
```

sets are unordered so indexing and slicing will not work.

In [37]:
```
st1[1]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-37-310081f4c4eb> in <module>
----> 1 st1[1]

TypeError: 'set' object is not subscriptable
```

In [ ]:
```

```

In [34]:
```
st1={1,"fan", 1500}
type(st1)
```

Out[34]:
```
set
```

In [40]:
```
s1=set("PythonP") # set function
print(s1)
```

```
{'y', 'h', 'P', 'o', 'n', 't'}
```

In [119]:
```
s2={[10,20],"SR",3} # list and dictionary can not be included in a set as they ar
print(s2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-119-809c5468098f> in <module>
----> 1 s2={[10,20],"SR",3}
      2 print(s2)

TypeError: unhashable type: 'list'
```

In [120]:
```
s2={(10,20),"SR",3} # tuple can be included in a set
print(s2)
```

```
{'SR', 3, (10, 20)}
```

In [42]:
```
s={10,20,30,20,50}
print(s)
s.update([40,60])
print(s)
s.remove(50)
print(s)
```

```
{10, 20, 50, 30}
{40, 10, 50, 20, 60, 30}
{40, 10, 20, 60, 30}
```

frozenset Datatype

set can be modified but frozen set can not be modified hence update() and remove () will not work on frozenset

In [43]:
```
fs=frozenset(s)
print(fs)
```

```
frozenset({20, 40, 10, 60, 30})
```

Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

In [ ]:
```

```

In [47]:
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
  File "<ipython-input-47-8299362d0095>", line 3
    "model": "Mustang","Abc",
                       ^
SyntaxError: invalid syntax
```

In [45]:
```
dict1={1:'A',2:'B',5:'D'}
print(dict1)
```

```
{1: 'A', 2: 'B', 5: 'D'}
```

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets

In [5]:
```
x = thisdict["model"]
print(x)
```

```
Mustang
```

In [6]:
```
x = thisdict.get("model") #same operation can be done using get() method
print(x)
```

```
Mustang
```

Change Values

You can change the value of a specific item by referring to its key name.

In [7]:
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

In [9]:
```
print(thisdict.values())
print(thisdict.keys())
```

```
dict_values(['Ford', 'Mustang', 2018])
dict_keys(['brand', 'model', 'year'])
```

In [10]:
```
print(len(thisdict))# length of dictionary
```

```
3
```

Adding Items

In [12]:
```
thisdict["color"] = "red"
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

*Removing Items

In [13]:
```
thisdict.pop("model") #pop() method removes the item with the specified key name
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964, 'color': 'red'}
```

In [14]:
```
thisdict.popitem() #popitem() method removes the last inserted item
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

In [15]:
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"] #del keyword removes the item with the specified key name
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

In [48]:
```
del thisdict #del keyword can also delete the dictionary completely
print(thisdict) #this will cause an error because "thisdict" no longer exists.
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-48-cd6397090ef5> in <module>
      1 del thisdict #del keyword can also delete the dictionary completely
----> 2 print(thisdict) #this will cause an error because "thisdict" no longer
 exists.

NameError: name 'thisdict' is not defined
```

In [16]:
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
thisdict.clear() #clear() method empties the dictionary
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{}
```

Copy a Dictionary

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

In [51]:
```
thisdict={
    1:"Ram",
    2:"Kishan",
    3:"Radha",
    4:"Raja"
}
print(thisdict)
mydict = thisdict.copy()
print(mydict)
thisdict[5]="Kala"
print(thisdict)
print(mydict)
```

```
{1: 'Ram', 2: 'Kishan', 3: 'Radha', 4: 'Raja'}
{1: 'Ram', 2: 'Kishan', 3: 'Radha', 4: 'Raja'}
{1: 'Ram', 2: 'Kishan', 3: 'Radha', 4: 'Raja', 5: 'Kala'}
{1: 'Ram', 2: 'Kishan', 3: 'Radha', 4: 'Raja'}
```

In [ ]:
```
```

In [22]:
```
mydict1 = dict(thisdict) #dict() function can also be used to make a copy of a di
print(mydict1)
```

```
{1: 'Ram', 2: 'Kishan', 3: 'Radha', 4: 'Raja', 5: 'Kala'}
```

create dictionary using list

In [5]:
```
countries=['USA', 'India', 'Germany', 'France']
cities=['Washington', 'New Delhi', 'Berlin', 'Paris']
z=zip(countries, cities)
d=dict(z)
print(d)
```

```
{'USA': 'Washington', 'India': 'New Delhi', 'Germany': 'Berlin', 'France': 'Par
is'}
```