# python

Bechir Brahem

## 1 variables

**python is dynamically typed** (variables are checked during run time) so no need to mention their types this can be achieved because every python variable is a reference to an object (+ or -).

```
a=4**0.5
a="degla"  #no errors
print a #a is "degla"
```

**Python is strong-typed** (in expressions variables are bound to their specific type)

```
temp ='Hello World!'
#temp = temp + 10; program terminates
#TypeError: cannot concatenate str and int objects
```

' and " are treated the same

```
"hello" , " 'abc " , 'abc'# are allowed
#"hello' is not allowed
```

## 2 arithmetic expressions

```
a=1/4# (a is 0)
a=1.0/4# (a is 0.25)
a=5.0//2# (a is 2.0)
a=9.0//3.1# (a is 2.0)
a=9.5//3.1# (a is 3.0)
```

if no one is float then it is integer division.

```
a=2**5# (a is 32)
a= 4**0.5# (a is 2.0)
```

** is exponentiation and can be used with irrationals

## 3 Strings

There are several ways to format strings in Python to interpolate variables. The new way (new in Python 3.6+) **F-Strings**

```
    x=10
    formatted = f"i have {x} bbbananas"# formatted = "i have 10 bbbananas"
```

the tried and true way(python2 -¿ 3.5) .format method

```
x=10
formatted="i have {} bbbananas".format(10)
```

# 4 common functions

insert, append, extend, clear, reverse, sort, count, join

```
    x=[1,2,3,4]
    x.insert(2,"hi") # [1,2,"hi",3,4]
    x.append(8) # [1,2,"hi",3,4,8]
    x,y=[1,2] , [3,4]
    x.append(y) # x=[1,2,[3,4]]
    x,y=[1,2] , [3,4]
    x.extend(y) # x=[1,2,3,4]
    x.clear() # x=[] removes all items from the list
    x=[1,2,3,4]
    x.reverse() # x=[4,3,2,1]
    x.sort() # x=[1,2,3,4]
    x=[1,2,3,4,3,2,1,4,10,2]
    x.count(2) # 3
    x.count(69) # 0
    x=["hi","my","name","is","Alfred"]
    " ".join(x) # "my name is Alfred"
```

**pop:**

- removes the item at the given position in the list and reutrn it.
- if no index is specified, removes and returns the last item

**remove.**

- removes the first item from the list with the given value
- throws ValueError if such value doesn't exist

**index.**

- returns the first index of the specified value
- returns ValueError if such value doesn't exist

# 5 slices

[start(included), end(excluded), step]

```
    x=[1,2,3,4,5,6,7]
    x[1:3]=['a','b','c'] # [1,'a','b','c',4,5]
    x=[1,2,3,4,5]
    x[::2] # [1,3,5,7]
    x[::-1]# [7,6,5,4,3,2,1]
```

# 6 list comprehension

```
[num*10 for num in range(1,6)] # [10,20,30,40,50]

[bool(val) for val in [0, [], '' ]] # [False,False,False]

items=["aaa", "nnn", "qqqq"]
[item[0]+item[1:] for item in items]
```

```
numbers = [1,2,3,4,5,6]
evens=[x for x in numbers if x %2==0]
odds=[x for x in numbers if x%2==1]
```

```
[x*2 if x%2==0 else x/2 for x in numbers]
[0.5, 4, 1.5, 8, 2.5, 12]
```

```
tmp= "This is so much fun!"
''.join( y for y in tmp if y not in "aeiou" )
# "Ths s s mch fn!"
```

# 7 Data Types.

## 7.1 Dictionnaries

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pairictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair **keys can't be repeated and must be immutable.**

```
tmp_dictionnary = dcit(key = 'value') #{'key': 'value'}
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
Dict[1] # "Geeks"
Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}
Dict[1] # [1,2,3,4]
```

**accessing all values in a dict**

```
for value in Dict.values():
  print(value)
#"Geeks",[1,2,3,4]
```

**accessing all keys in a dict**

```
for key in Dict.keys():
  print(key)
#"Name", 1
```

**for both**

```
x={(1, 2): [1, 2, 3], 1: 'ax', 'alala': 12}
for key,value in x.items():
  print(key,value)

#((1,2),[1,2,3])
#(1,"ax")
#("alala",12)
```

**check existence.**

```
x={1:"pp", "name":69}

1 in x #True
69 in x # False

69 in x.values() # True
[1,2] in x.values() # False
```

if the key value already exists, the value gets updated otherwise a new Key with the value is added to the Dictionary.

```
d={}
d[1]="aa" # d is {1:"aa"}
d[(1,2)]={"key":"oop"} # d is {1:"aa", (1,2):{"key","oop"}}
d[(1,2)]=2 # d is {1:"aa", (1,2):2}
```

**copy**

```
d=dict(a=1,b=2,c=3)
c=d.copy()
c is d # False
c==d # True
```

**get** retrieves a key in an object and return None instead of a KeyError if the key doesn't exist

```
d= dict(a=1,b=2,c=3)
d['a'] # 1
d.get('a') # 1
d["pp"] # KeyError
d.get("pp") # None
```

**fromkeys** Creates key-value pairs from comma seperated values and it has no effect on an already created dict:

```
{}.fromkeys("a","b") # {'a':'b'}
{}.fromkeys(["email"],"unknown") # {'email': 'unknown'}
{}.fromkeys("a",[1,2,3,4,5]) # {"a":[1,2,3,4,5]}
a={1:"pp"}
a.fromkeys(1,"po") # a is still {1:"pp"}
```

**pop** takes a single argument corresponding to a key and removes the key:value pair. returns the value associated with the key.

```
d={1:"q":"b":3}
d.pop("b") # 3 d is {1:"q"}
d.pop() # TypeError
d.pop("o")  # KeyError
```

**update** update keys and values in a dict with another dict

```
first=dict(a=1,b=2,c=3)
second={1:"alfred big"}
second.update(first)
second # {"a":1, "b":2, "c":3, 1:"alfred big"}
```