```
print "Welcome to Python!"

#Single line Comment
my_int = 7
my_float = 1.23
my_bool = True
print my_int

""" Multi line comment
"""
def spam():
        eggs=12
        return eggs
print spam()

#Adding two numbers
a=5
b=7
sum=a+b
print sum

#exponential
eggs=10**2
print eggs

spam=5%4
print spam

python=1.234
monty_python=python**2
print monty_python

# Strings

brian="hello life!"
print brian

brian='This isn\'t flying, this is falling with style!'
print brian

#Printing 5th letter

fifth_letter="MONTY"[4]
```

```
print fifth_letter

parrot="Norwegian Blue"
print len(parrot)
print "Norwegian Blue".lower()
print parrot.upper()

#convert to string
pi=3.14
print str(pi)

#String concatenation

print "Spam " + "and" + " eggs"

#Turn 3.14 into string

print "The value of pi is around " + str(3.14)

# Using %s in print statement
string_1 = "Gautham"
string_2 = "Chennai"

print "My name is %s. I live in %s. " %(string_1,string_2)

name = raw_input("What is your name?")
quest = raw_input("What is your quest?")
color = raw_input("What is your favorite color?")

print "Ah, so your name is %s, your quest is %s, " \
"and your favorite color is %s." % (name, quest, color)

#Date
from datetime import datetime
now=datetime.now()
print now
print now.year
print now.month
print now.day
print '%s%s%s' % (now.month, now.day, now.year)
print '%s:%s:%s' %(now.hour,now.minute,now.second)

#if else
```

```
def clinic():
    place = raw_input ("Where are you living ?").lower()

    if place=="tiruppur" :
        native= raw_input("what is special in Tiruppur ?").lower()
        if native==" " :
            print "please say something about Tiruppur"
            clinic()
        if native=="":
            print "empty"
        else:
            print native
            print "Nice to hear from you !!! "
    else:
        native=raw_input("what is special in Chennai ?").lower()
        if native==" " :
            print "please say something about Chennai"
            clinic()
        else:
            print "Nice to hear from you !!! "

clinic()
```

#Boolean

```
bool_one = (2 <= 2) and "Alpha" == "Bravo"  # We did this one for you!
print bool_one
```

#Function

```
def using_control_once():
    if 3>2:
        return "Success #1"

def using_control_again():
    if 5==5:
        return "Success #2"

print using_control_once()
print using_control_again()

#else if
def greater_less_equal_5(answer):
    if answer>5:
        return 1
    elif answer<5:
        return -1
    else:
        return 0

print greater_less_equal_5(4)
```

# converting 'Python'->' ythonPay'

```
input= raw_input("Enter the String").lower()
x=1
first_letter=input[0]
result=""
for letter in input:     # First Example
   if x==1 :
       x+=1
   else:
       current_letter=letter
       result=result + current_letter
result=result + input[0] + "ay"
print result
```

#Checking for empty input

```
original=raw_input("Enter a string")
if len(original)>0:
    print original
else:
    print "empty"
```

#Slicing

```
pyg = 'ay'

first = "sachins"
new_word = first[1:len(first)] + pyg
print new_word
```

#Sending a value to Function

```
def tax(bill):
    """Adds 8% tax to a restaurant bill."""
    bill *= 1.08
    print "With tax: %f" % bill
    return bill
```

```
def tip(bill):
    """Adds 15% tip to a restaurant bill."""
    bill *= 1.15
    print "With tip: %f" % bill
    return bill

meal_cost = 100
meal_with_tax = tax(meal_cost)
meal_with_tip = tip(meal_with_tax)

#Calling a function from a function
def one_good_turn(n):
    return n + 1

def deserves_another(n):
    return one_good_turn(n) + 2

print deserves_another(5)

#finding max, min, abs
def biggest_number(*args):
    print max(args)
    return max(args)

def smallest_number(*args):
    print min(args)
    return min(args)

def distance_from_zero(arg):
    print abs(arg)
    return abs(arg)


biggest_number(-10, -5, 5, 10)
smallest_number(-10, -5, 5, 10)
distance_from_zero(-10)

#Exponential function
def power(base,exponent):  # Add your parameters here!
    result = base**exponent
    print "%d to the power of %d is %d." % (base, exponent, result)
```

```
power(37,4)

#Finding type of an integer, float, string

print type(42)

print type(4.2)
print type('spam')

#Built in math function
import math
print math.sqrt(25)

#using 'or' in conditional statement
def distance_from_zero(p):
    if type(p)==int or type(p)==float:
        return abs(p)
    else:
        return "Nope"
distance_from_zero(4)

#Functions
def trip_cost(city,days,spending_money):
    print rental_car_cost(days) + hotel_cost(days) + plane_ride_cost(city) + spending_money
    return rental_car_cost(days) + hotel_cost(days) + plane_ride_cost(city) + spending_money


def rental_car_cost(days):
    cost=40
    pay=cost*days

    if days>=7:
        finalpay=pay-50
        return finalpay
    elif days>=3:
        finalpay=pay-20
        return finalpay
    else:
        finalpay=pay
        return finalpay

def hotel_cost(nights):
    return 140 * nights
```

```
def plane_ride_cost(city):
    if city=="Charlotte":
        return 183
    elif city=="Tampa":
        return 220
    elif city=="Pittsburgh":
        return 222
    elif city=="Los Angeles":
        return 475

trip_cost("Los Angeles",5,600)

#range() is a shortcut for generating a list

print range(5) # [0,1,2,3,4]
print range(1,6) # [1,2,3,4,5]
print range(1,6,3) #[1,4]

#Using range function
n = [3, 5, 7]

def print_list (x):
    for i in range(0, len(x)):
        print x[i]

print_list(n)

#List
zoo_animals = ["pangolin", "cassowary", "sloth","lion" ];

if len(zoo_animals) > 3:
        print "The first animal at the zoo is the " + zoo_animals[0]
        print "The second animal at the zoo is the " + zoo_animals[1]
        print "The third animal at the zoo is the " + zoo_animals[2]
        print "The fourth animal at the zoo is the " + zoo_animals[3]

#Adding two numbers from the list
numbers = [5, 6, 7, 8]
print numbers[1] + numbers[3]

#Replacing a value in the list
```

```
zoo_animals = ["pangolin", "cassowary", "sloth", "tiger"]
zoo_animals[2] = "hyena"
zoo_animals[3]="lion"

#Appending
suitcase = []
suitcase.append("sunglasses")
suitcase.append("coolers")
suitcase.append("cricketer")
suitcase.append("nice")


list_length = len(suitcase) # Set this to the length of suitcase

print "There are %d items in the suitcase." % (list_length)
print suitcase

#Slicing in the List
suitcase = ["sunglasses", "hat", "passport", "laptop", "suit", "shoes"]
first  = suitcase[0:2]  # The first and second items (index zero and one)

animals = "catdogfrog"
cat  = animals[:3]   # The first three characters of animals

#Finding Index and inserting a element
animals = ["aardvark", "badger", "duck", "emu", "fennec fox"]
duck_index=animals.index("duck")
print duck_index

animals.insert(duck_index,"cobra")
print animals

#Removing a element from the list
backpack = ['xylophone', 'dagger', 'tent', 'bread loaf']
backpack.remove('dagger')

#Removing the first item in the list
n = [1, 3, 5]
n.pop(0)
print n


#Accessing every element in the list
```

```
my_list = [1,9,3,8,5,7]

for number in my_list:
    print 2 * number

#Sorting function in the List
start_list = [5, 3, 1, 2, 4]
square_list = []

for x in start_list:
    square_list.append(x ** 2)

square_list.sort()
print square_list

#Sending the List as a function parameter
def fizz_count(x):
    count = 0
    for text in x:
        if text == "fizz":
            count = count + 1
    return count

lost = ["fizz","fizz", "gautham", "note"]
small = fizz_count(lost)
print small

#Modifying the list value from function
n = [3, 5, 7]
# Add your function here
def list_extender(lst):
    lst.append(9)
    return lst

print list_extender(n)

# Passing two list as two argument
m = [1, 2, 3]
n = [4, 5, 6]

def join_lists(x,y):
    return x + y # [1, 2, 3, 4, 5, 6]
```

```
print join_lists(m, n)
```

#List of List
```
n = [[1, 2, 3], [4, 5, 6, 7, 8, 9]]

def flatten(lists):
    result=[]
    for lst in lists:
        for numbers in lst:
            result.append(numbers)
    return result    # [1,2,3,4,5,6,7,8,9]

print flatten(n)
```

#Building a List
```
evens_to_50 = [i for i in range(51) if i % 2 == 0]
print evens_to_50
```

#List slicing
```
l = [i ** 2 for i in range(1, 11)] # Should be [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
print l[2:9:2] # [9,25,49,81]
```

#Dictionary
```
residents = {'Puffin' : 104, 'Sloth' : 105, 'Burmese Python' : 106}
print residents['Puffin'] # Prints Puffin's room number
print residents.items() // [
```

#Reversing the List using list slicing

```
my_list = range(1, 11) # [1,2,3,4,5,6,7,8,9,10]

backwards=my_list[::-1]
print backwards
```

#Accessing keys and values separately
```
my_dict={"sachin":100,"ganguly":70,"dravid":110}
print my_dict.keys()
print my_dict.values()

for keys in my_dict:
    print keys,my_dict[keys]
```

#Empty dictionary and adding elements to it

```
menu = {}
menu['Chicken Alfredo'] = 14.50 # Adding new key-value pair
print "There are " + str(len(menu)) + " items on the menu."
print menu

#Deleting an element
zoo_animals = { 'Unicorn' : 'Cotton Candy House',
'Sloth' : 'Rainforest Exhibit',
'Bengal Tiger' : 'Jungle House',
'Atlantic Puffin' : 'Arctic Exhibit',
'Rockhopper Penguin' : 'Arctic Exhibit'} # A dictionary (or list) declaration may break across
multiple lines

del zoo_animals['Unicorn']
print zoo_animals

#Adding the list as a value in the dictionary
inventory = {
    'gold' : 500,
    'pouch' : ['flint', 'twine', 'gemstone'], # Assigned a new list to 'pouch' key
    'backpack' : ['xylophone','dagger', 'bedroll','bread loaf']
}

# Adding a key 'burlap bag' and assigning a list to it
inventory['burlap bag'] = ['apple', 'small ruby', 'three-toed sloth']

# Sorting the list found under the key 'pouch'
inventory['pouch'].sort()
inventory['pocket'] = ['seashell','strange berry','lint']
inventory['backpack'].sort()
inventory['backpack'].remove('dagger')
inventory['gold']=50+500

#Accessing every key in the dictionary

prices ={
    "apple":2,
    "banana":4,
    "orange":1.5,
    "pear":3
    }
stock ={
    "banana":6,
    "apple":0,
    "orange":32,
    "pear":15
    }

for x in prices:
    print x
    print "price: %s" % prices[x]
    print "stock: %s"  % stock[x]
```

#String looping

```
for letter in "Codecademy":
    print letter
```

#A List contains three dictionary as a value

```
lloyd = {
    "name": "Lloyd",
    "homework": [90.0, 97.0, 75.0, 92.0],
    "quizzes": [88.0, 40.0, 94.0],
    "tests": [ 75.0, 90.0]
}
alice = {
    "name": "Alice",
    "homework": [100.0, 92.0, 98.0, 100.0],
    "quizzes": [82.0, 83.0, 91.0],
    "tests": [89.0, 97.0]
}
tyler = {
    "name": "Tyler",
    "homework": [0.0, 87.0, 75.0, 22.0],
    "quizzes": [0.0, 75.0, 78.0],
    "tests": [100.0, 100.0]
}
students=[lloyd,alice,tyler]
```

#Accesing dictionary elements from the list

```
lloyd = {
    "name": "Lloyd",
    "homework": [90.0, 97.0, 75.0, 92.0],
    "quizzes": [88.0, 40.0, 94.0],
    "tests": [ 75.0, 90.0]
}
alice = {
    "name": "Alice",
    "homework": [100.0, 92.0, 98.0, 100.0],
    "quizzes": [82.0, 83.0, 91.0],
    "tests": [89.0, 97.0]
}
tyler = {
    "name": "Tyler",
    "homework": [0.0, 87.0, 75.0, 22.0],
    "quizzes": [0.0, 75.0, 78.0],
    "tests": [100.0, 100.0]
}
students=[lloyd,alice,tyler]

for x in students:
    for y in x:
        print x[y]
```

#While Loop

```
count =0
while count < 10:
    print "Hello, I am a while and count is", count
    count += 1
```

#while loop with 'break' statement

```
count = 0

while True:
    print count
    count += 1
    if count >= 10:
        break
```

#while/else statement

```
import random
count = 0
while count < 3:
    num = random.randint(1, 6)
    print num
    if num == 5:
        print "Sorry, you lose!"
        break
    count += 1
else:
    print "You win!"
```

#for loop

```
d = {'a': 'apple', 'b': 'berry', 'c': 'cherry'}

for key in d:
    print key +  " " +  d[key]
```

#Iterating multiple lists

```
list_a = [3, 9, 17, 15, 19]
list_b = [2, 4, 8, 10, 30, 40, 50, 60, 70, 80, 90]
list_c = [ 100,101,102,103,104,105,106]
larger=list_a[0]
for a, b,c in zip(list_a, list_b ,list_c):
    # Add your code here!
    if a>b:
        larger=a
    else:
        larger=b
    print larger
```

zip(list_a, list_b ,list_c) -> [(3, 2, 100), (9, 4, 101), (17, 8, 102), (15, 10, 103), (19, 30, 104)]

#for/else loop

```
fruits = ['banana', 'apple', 'orange', 'tomato', 'pear', 'grape']

print 'You have...'
for f in fruits:
    if f == 'tomato':
        print 'A tomato is not a fruit!' # (It actually is.)
        break
    print 'A', f
else:
    print 'A fine selection of fruits!'
```

#Anonymous function

```
my_list = range(16)
print filter(lambda x: x % 3 == 0, my_list)
```

#hidden message
garbled = "IXXX aXXmX aXXXnXoXXXXXtXhXeXXXXrX sXXXXeXcXXXrXeXt mXXeXsXXXsXaXXXXXXgXeX!XX"
message= **filter(lambda x: x!="X" , garbled)**
print message

#Base2 number system
print 0b1,   #1
print 0b10,   #2
print 0b11,   #3
print 0b100, #4
print 0b1 + 0b11
print 0b11 * 0b11

print bin(1) #0b1

print int("1",2) #1
print int("10",2) #2
print int("111",2) #7
print int(bin(5),2) #5

#Shift operator

shift_right= 0b1100 >>2
shift_left= 0b1 <<2
print shift_right #3
print shift_left #4

print bin(0b1110 & 0b101) # 0b100
print 0b1110 & 0b101 #4

print bin (0b1110 | 0b101) #0b1111
print 0b1110 | 0b101 #15

#Shift right and Shift left operator

shift_right = 0b1100
shift_left = 0b1

shift_right = 0b1100
shift_left = 0b1

shift_right = shift_right >> 2

```
shift_left = shift_left << 2
print bin(shift_right)
print bin(shift_left)

#XOR operator is used to flip bits
a = 0b11101110
mask=0b11111111
print bin(a ^ mask)
```

#Python Writing to a file

```
my_list = [i**2 for i in range(1,11)]

f = open("output.txt", "w") # r+ is for read and write option

for item in my_list:
    f.write(str(item) + "\n")

f.close()
```

#Python reading a file

```
my_file = open("output.txt","r")

print my_file.read()

my_file.close()
```

#Reading every line

```
my_file=open("text.txt","r")
for line in my_file:
    print line
my_file.close()
```

# Using 'with'

```
with open("text.txt", "w") as textfile:
    textfile.write("Success!")
    textfile.close
```

#To check if file is closed

```
with open("text.txt","w") as my_file:
    my_file.write("Success story !!!")
```

```
if my_file.closed:
    print my_file.closed
    my_file.close()
```

#creating a file

```
import io
with io.FileIO("./document/my_bag.txt", "w") as file:
    file.write("hello")
```

Pdb - python Interactive Debugger:

pdb implements an interactive debugging environment for Python programs. It includes features to let you pause your program, look at the values of variables, and watch program execution step-by-step, so you can understand what your program actually does and find bugs in the logic.

From the command line:

Running the debugger from the command line causes it to load your source file and stop execution on the first statement it finds.

```
python -m pdb pdb_script.py
```

pdb.set_trace() is just a Python function, so you can call it at any point in your program. This lets you enter the debugger based on conditions inside your program, including from an exception handler or via a specific branch of a control statement.

Ref:
https://pymotw.com/2/pdb/
https://pythonconquerstheuniverse.wordpress.com/2009/09/10/debugging-in-python/
http://www.onlamp.com/pub/a/python/2005/09/01/debugger.html
https://docs.python.org/2/library/pdb.html

Ipython

IPython is an interactive shell for the Python programming language that offers enhanced introspection, additional shell syntax, tab completion and rich History

run - to execute a python file

edit - to open a editor in ipython

colors LightBG , %colors nocolor

clear - to clear the screen

doctest_mode -

Toggle doctest mode on and off.

This mode is intended to make IPython behave as much as possible like a plain Python shell, from the perspective of how its prompts, exceptions and output look.

who

Print all interactive variables, with some minimal formatting.

whos

Like %who, but gives some extra information about each variable.

history

IPython stores both the commands you enter, and the results it produces. You can easily go through previous commands with the up- and down-arrow keys, or access your history in more sophisticated ways.

dhist

Print your history of visited directories.

dirs

Return the current directory stack.

debug

Activate the interactive debugger.

env

Get,set or list environment variables

pwd, cd

alias

Ref :

https://ipython.org/ipython-doc/3/interactive/magics.html#magic-colors

Built In functions:

all()	Return True if all elements of the list are true (or if the list is empty).
any()	Return True if any element of the list is true. If the list is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of list as a tuple.
len()	Return the length (the number of items) in the list.
list()	Convert an iterable (tuple, string, set, dictionary) to a list.
max()	Return the largest item in the list.
min()	Return the smallest item in the list

sorted()	Return a new sorted list (does not sort the list itself).
	
	Ex:
	pySet = {'e', 'a', 'u', 'o', 'i'}
	print(sorted(pySet, reverse=True))

sum()	Return the sum of all elements in the list.

List functions:

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

Elegant way to create a list:

Ex:

```
pow2 = [2 ** x for x in range(10)]
```

```
[x+y for x in ['Python ','C '] for y in ['Language','Programming']]
```

Ex:

```
for fruit in ['apple','banana','mango']:
   print("I like",fruit)
```

We can test if an item present in list or not

```
my_list = ['p','r','o','b','l','e','m']
```

```
print('p' in my_list)
```

Zip function:

The zip function iterates through multiple iterables, and aggregates them. Consider you have two lists, and you instead want them to be one list, where elements from the shared index are together.

```
x = [1,2,3,4]
y = [7,8,3,2]
z = ['a','b','c','d']

for a,b,c in zip(x,y,z):

    print(a,b,c)
```

Output:

```
1 7 a
2 8 b
3 3 c
4 2 d
```

```
print(zip(x,y,z)) // creates zip object

print(list(zip(x,y,z))) // converts into a list which prints [(1, 7,
'a'), (2, 8, 'b'), (3, 3, 'c'), (4, 2, 'd')]
```

Sets

A set is an unordered collection of distinct elements. As a simple example, consider the following:

```
animals = {'cat', 'dog'}
print('cat' in animals)   # Check if an element is in a set; prints "True"
print('fish' in animals)  # prints "False"
animals.add('fish')       # Add an element to a set
animals.remove('cat')     # Remove an element from a set
print(len(animals))       # Prints "2"
```

Yield:

```
CORRECT_COMBO = (3, 6, 1)

def combo_gen():
    for c1 in range(10):
        for c2 in range(10):
            for c3 in range(10):
                yield (c1, c2, c3)

for (c1, c2, c3) in combo_gen():
    print(c1, c2, c3)
    if (c1, c2, c3) == CORRECT_COMBO:
        print('Found the combo:{}'.format((c1, c2, c3)))
        Break
```

Iterables:

When you create a list, you can read its items one by one. Reading its items one by one is called iteration:

```
mylist = [1, 2, 3]
for i in mylist:
   print(i)
```

mylist is an *iterable*. When you use a list comprehension, you create a list, and so an iterable:

These iterables are handy because you can read them as much as you wish, but you store all the values in memory and this is not always what you want when you have a lot of values.

Generators:

Generators are iterators, but you can only iterate over them once. It's because they do not store all the values in memory, they generate the values on the fly:

```
mygenerator = (x*x for x in range(3))

for i in mygenerator:

   print(i)
```

It is just the same except you used () instead of []. BUT, you **cannot** perform for i in mygenerator a second time since generators can only be used once: they calculate 0, then forget about it and calculate 1, and end calculating 4, one by one.

--
-

Classes and Objects:

Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects.

Constructors in Python:

Class functions that begins with double underscore (__) are called special functions as they have special meaning.

Of one particular interest is the __init__() function. This special function gets called whenever a new object of that class is instantiated.

This type of function is also called constructors in Object Oriented Programming (OOP). We normally use it to initialize all the variables.

Example program:

```
class ComplexNumber:
    def __init__(self,r = 0,i = 0):
        self.real = r
        self.imag = i
    def getData(self):
        print(self.real,self.imag)

c1 = ComplexNumber(2,3) //  Create a new ComplexNumber object
c1.getData() //  Call getData() function

c2 = ComplexNumber(5) //  Create another ComplexNumber object
c2.attr = 10 // create a new attribute 'attr'
print((c2.real, c2.imag, c2.attr)) //  Output: (5, 0, 10)
```

Inheritance:

```
class Person(object):
        def __init__(self,first_name,last_name):
                self.first=first_name
                self.last=last_name

        def name(self):
                return self.first + " "      + self.last

class Employee(Person):
        def __init__(self,first,last,emp_id):
                super(Employee, self).__init__(first,last)
                self.employee_id=emp_id

        def employee_info(self):
                print self.name()
                print self.employee_id

x = Employee("gautham","sundar",425)
```

x.employee_info()

Deleting Attributes and Objects

Any attribute of an object can be deleted anytime, using the del statement.
del c2.attr

Ref : https://www.programiz.com/python-programming/class

Decorator:

Python's decorators allow you to extend and modify the behavior of a callable (functions, methods, and classes) *without* permanently modifying the callable itself.

Code:

```
def smart_divide(func):
   def inner(a,b):
      print "I am going to divide",a,"and",b
      if b == 0:
         print("Whoops! cannot divide")
         return

      return func(a,b)
   return inner

@smart_divide
def divide(a,b):
            return a/b

print divide(4,2)
```

Ref :
https://www.google.com/url?q=https%3A%2F%2Fdbader.org%2Fblog%2Fpython-decorators&sa=D&sntz=1&usg=AFQjCNHBYpWYba4raQdwAw1sJgDuNB_RPQ

https://www.programiz.com/python-programming/decorator

-

How to use *args and **kwargs in Python

The special syntax, *args and **kwargs in function definitions is used to pass a variable number of arguments to a function.

The single asterisk form (*args) is used to pass a *non-keyworded*, variable-length argument list, and the double asterisk form is used to pass a *keyworded*, variable-length argument list.

Code:

```
def test_var_args(farg, *args):
    print "formal arg:", farg
    for arg in args:
        print "another arg:", arg


test_var_args(1, "two", 3)
```

Result:

```
formal arg: 1
another arg: two
another arg: 3
```

Here is an example of how to use the keyworded form. Again, one formal argument and two keyworded variable arguments are passed.

```
def test_var_kwargs(farg, **kwargs):
    print "formal arg:", farg
    for key in kwargs:
        print "another keyword arg: %s: %s" % (key, kwargs[key])


test_var_kwargs(farg=1, myarg2="two", myarg3=3)
```

Result:

```
formal arg: 1
another keyword arg: myarg2: two
another keyword arg: myarg3: 3
```

Ref: https://www.saltycrane.com/blog/2008/01/how-to-use-args-and-kwargs-in-python/

----

lambda():

The lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name. These functions are throw-away functions, i.e. they are just needed where they have been created. Lambda functions are mainly used in combination with the functions filter(), map() and reduce()

----

map():

One of the common things we do with list and other sequences is applying an operation to each item and collect the result.
For example, updating all the items in a list can be done easily with a **for** loop:

```
items = [1, 2, 3, 4, 5]
squared = []
for x in items:
        squared.append(x ** 2)
```

Since this is such a common operation, actually, we have a built-in feature that does most of the work for us.

The **map(aFunction, aSequence)** function applies a passed-in function to each item in an iterable object and returns a list containing all the function call results.

```
def sqr(x): return x ** 2

list(map(sqr,items) )
```

It also has some performance benefit because it is usually faster than a manually coded **for** loop. On top of those, **map** can be used in more advance way. For example, given multiple sequence arguments, it sends items taken form sequences in parallel as distinct arguments to the function:
Example:

my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

- print **map**(lambda x : x%3 , my_list)

o/p: [0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0]

http://www.bogotobogo.com/python/python_fncs_map_filter_reduce.php

----
-

filter():

The function filter(f,l) needs a function f as its first argument. f returns a Boolean value, i.e. either True or False. This function will be applied to every element of the list *l*. Only if f returns True will the element of the list be included in the result list.

Example:

my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

- print **filter**(lambda x : x%3 , my_list)

[1, 2, 4, 5, 7, 8, 10, 11, 13, 14]

-

split():

x = 'blue,red,green'

- x.split()

o/p: ['blue,red,green']

- x.**split**(',')

o/p: ['blue', 'red', 'green']

-

join():

```
seq = ("a", "b", "c");
print “-”.join( seq )
```

o/p: a-b-c

Ref : https://www.programiz.com/python-programming/methods/string/join

----

Numpy:

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Cache and cookies:

Cache and cookies are two different forms of temporary storage kept on client's machine to improve the performance of web pages.

Cookies:

Cookie is a very small piece of information that is stored on the client’s machine by the web site and is sent back to the server each time a page is requested.

Cookies was first introduced by Netscape. Cookies are usually used to store information needed for shorter periods. At the end of this, a cookie becomes expired.

Cache:

Cache is a temporary storage of web page resources stored on client’s machine for quicker loading of the web pages.

When you open some websites with large pictures and video’s, it might take a considerable amount of time for the website to load. The web browser stores the site contents like the images, videos, audio etc on your computer so the next time you load the same website you will find it loading faster.

What is the difference between Cookies and Caches:

1. Although cookies and cache are two ways to store data on client's machine, they serve different purposes.
2. Cookie is used to store information to track different characteristics related to user, while cache is used to make the loading of web pages faster.
3. Cookies stores information such as user preferences, while cache will keep resource files such as audio, video or flash files.
4. Typically, cookies expire after some time, but cache is kept in the client's machine until they are removed manually by the user.