CryptX + Recognizance: Basics of Pythor

Starters

In [1]:
```
# Comments
# This is a python tutorial and a single line comment
''' This is a multiline comment
    pretty awesome!!
    Let me introduce you to jennifer!'''
```

```
' This is a multiline comment\n    pretty awesome!!\n    Let me introduce you to jennifer!'
```

In [2]:
```
# Simple imports
import math
import random
```

In [6]:
```
print(math.factorial(5))
math.gcd(64,8)
```

```
120
```

```
8
```

In [8]:
```
help(math)
```

In [9]:
```
# importing specific functions from modules
# imports just the factorial function from math
from math import factorial

# imports all the functions from math
from math import *
```

In [10]:
```
factorial(25)
```

```
15511210043330985984000000
```

In [13]:
```
# Giving aliases
# The Module name is alaised
import math as m

# The function name is alaised
from math import factorial as fact
import numpy
```

In [14]:
```
numpy.array([1,2])
```

```
array([1, 2])
```

In [15]:
```
import numpy
```

In [17]:
```
from math.factorial(5)
```

```
  File "<ipython-input-17-629147b0724b>", line 1
    from math.factorial(5)
                        ^
SyntaxError: invalid syntax
```

In [18]:
```
# Calling imported functions
# If you import the module you have to call the functions from the module
import math
print (math.factorial(12))

# If you import the functions you can call the function as if it is in your progr
from random import randrange as rg
print (rg(23, 1000))
```

```
479001600
550
```

In [19]:
```
# Variables
msg = "Python!"  # String
v2 = 'Python!'   # Also String works same
v1 = 2           # Numbers
v3 = 3.564       # Floats / Doubles
v4 = True        # Boolean (True / False)
```

In [20]:
```
# print()
# automatically adds a newline
print (msg)
print (v2)
print (type(v1))
print (v3)
print (v4)
print ("Hello Python!")
```

```
Python!
Python!
<class 'int'>
3.564
True
Hello Python!
```

In [21]:
```
# Note: Both " and ' can be used to make strings. And this flexibility allows for
msg2 = 'Jennifer said, "I love Python!"'
msg3 = 'After that Jennifer\'s Python Interpreter said it back to her!'
msg4 = 'Of Course she used the command `print("I love Jennifer")`'

print (msg2)
print (msg3)
print (msg4)
```

```
Jennifer said, "I love Python!"
After that Jennifer's Python Interpreter said it back to her!
Of Course she used the command `print("I love Jennifer")`
```

In [ ]:
```
# input()
msg = input()
```

In [1]:
```
# input() with message
msg = input ("Provide some input: ")
print (type(msg))
```

```
Provide some input: soe
<class 'str'>
```

In [2]:
```
# Check for specific input without storing it
if input("Enter something: ") == "something":
    print ("Something something")
else: print ("Not something")
```

```
Enter something: kujij
Not something
```

In [3]:
```
# Python takes every input as a string
# So, if required you can convert to the required type
msg = input("Enter a number: ")
print (type(msg))

msg = int(input ("Enter a number again, if not a number this will throw an error:
print (type(msg))
```

```
Enter a number: 89
<class 'str'>
Enter a number again, if not a number this will throw an error: 8
<class 'int'>
```

In [5]:
```
# Basic Arithmetic operations
# Add
print (3 + 2)
print (3.4565 + 56.232)
print ('------------')

# Subtract
print (3 - 4)
print (34.56 - 3.78)
print ('------------')

# Multiply
print (4 * 3)
print (7.56 * 34)
print ('------------')

# Division
print (5 / 2)
print (5.0 / 2)
print (5 / 2.0)
print (25.0 / 5)
print ('------------')

# Exponents
print (4 ** 4)
print (5.67 ** 3)
print ('------------')

# Modulo
print (10%3)
print (10//11)
```

```
5
59.6885
------------
-1
30.78
------------
12
257.03999999999996
------------
2.5
2.5
```

```
2.5
5.0
------------
256
182.28426299999998
------------
1
0
```

Conditionals

In [6]:
```
# if..else
v1 = 5
if v1 == 5:
    print (v1)
    v2 = 102
    print(v2)
else:
    print ("v1 is not 5")
```

```
5
102
```

In [7]:
```
# if..elif..else
s1 = "Jennifer"
s2 = "loves"
s3 = "Python"
if s1 == "Python":
    print ("s1 is Python")
elif s2 == "Jennifer":
    print ("s2 is Jennifer")
elif s1 == "loves":
    print ("s1 is loves")
else:
    print ("Jennifer loves Python!")
```

```
Jennifer loves Python!
```

In [8]:
```
# One liner
v1 = 6
x = 10 if v1 == 5 else 13
print (x)
```

```
13
```

In [9]:
```
# Let's see the conditionals available
v1 = "Jennifer"
v2 = "Python"
v3 = 45
v4 = 67
v5 = 45

# Test for equality
print (v1 == v2)

# Test for greater than and greater than equal
print (v4 > v3)
# print (v5 >= v2)

# Test for lesser than and lesser than equal
print (v4 < v3)
# print (v5 <= v2)

# Inequality
print (v1 != v2)
```

```
False
True
False
True
```

In [10]:
```
# Note:
v1 = 45
v2 = "45"
print (v1 == v2) # False
print (str(v1) == v2) # True
```

```
False
True
```

In [11]:
```
# Ignore case when comparing two strings
s1 = "JeNnifer"
s2 = "jennIfer"

print (s1 == s2) # False
print (s1.lower() == s2.lower()) # True
# OR
print (s1.upper() == s2.upper()) # True
```

```
False
True
True
```

In [12]:
```
# Checking multiple conditions 'and' and 'or'
v1 = "Jennifer"
v2 = "Python"

# 'and' -> evaluates true when both conditions are True
print (v1 == "Jennifer" and v2 == "Python")
# 'or' -> evaluates true when any one condition is True
print (v1 == "Python" or v2 == "Python")
```

```
True
True
```

> Note:
>
> When making comparisons with string with '>' or '<' The strings are compared lexog

In [13]:
```
s1 = "Pynnifer"
s2 = "Pethon"

print (s1 > s2) # True -> since 'Jennifer' comes lexographically before 'Python'
```

```
True
```

In [14]:
```
# Check whether a value is in a list -> 'in'
l1 = [23, 45, 67, "Jennifer", "Python", 'A']

print (23 in l1)
print ("A" in l1)
print ("Python" in l1)
print (32 in l1)
```

```
True
True
True
False
```

In [15]:
```
# Putting it together
l1 = [23, 1, 'A', "Jennifer", 9.34]

# This is True, so the other statements are not checked
if 23 in l1 and 'B' not in l1: # Note: use of 'not'
    print ("1")
elif 23 >= l1[0]: # True
    print ("2")
elif 2.45 < l1[-1]: # True
    print ("3")
```

```
1
```

In [16]:
```
# Checking if list is empty
l1 = []
l2 = ["Jennifer"]

if l1:
    print (1)
elif l2:
    print (2)
```

```
2
```

Loops

In [17]:
```
# Simple while
# Loop runs till the condition is True
v1 = 5
while v1 <= 10:
    print (v1)
    v1 += 1
```

```
5
6
7
8
9
10
```

In [18]:

```python
# Infinite Loops
while 1:
    print (1)

# while True:
#     print (1)
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

In [19]:

```python
# One Liner while
v1 = 0
while v1 <= 40:
    v1 += 1
print (v1)
```

```
41
```

In [ ]:

In [20]:
```
# Terminate loop on a certain user input
# Note: The loop will break only when the user inputs 100
v1 = 1
while v1 != 100:
    v1 = int(input("Enter new v1: "))
    print ("v1 modified to: " + str(v1))
```

```
Enter new v1: 787
v1 modified to: 787
Enter new v1: 13
v1 modified to: 13
Enter new v1: 100
v1 modified to: 100
```

In [ ]:
```
# 'break' -> breaks out of loop, doesn't execute any statement after it
while 1:
    v1 = int(input())
    if v1 == 100:
        break
    print (v1)
```

In [ ]:
```
# 'continue' -> continues to next iteration, skips all statements after it for th
# Note: When 'v1' < 100 the last print statement is skipped and the control moves
while 1:
    print ("Iteration begins")
    v1 = int(input())
    if v1 == 100:
        break
    elif v1 < 100:
        print ("v1 less than 100")
        continue
print ("Iteration complete")
```

In [21]:
```
# while with lists
l1 = ["Jennifer", 12, "Python", 'A', 56, 'B', 2.12, "Scarlett"]
l2 = []
i = 0
while i < len(l1):
    if type(l1[i]) == int:
        l2.append(l1[i])
    i += 1
print (l2)
```

```
[12, 56]
```

In [22]:
```
# Removing all instances of a specific value in list
l1 = ['A', 'B', 'C', 'D', 'A', 'E', 'Q', 'A', 'Z', 'A', 'Q', 'D', 'A']
while 'A' in l1: l1.remove('A')
print (l1)
```

```
['B', 'C', 'D', 'E', 'Q', 'Z', 'Q', 'D']
```

In [ ]:
```
# Filing a dictionary
d1 = {}
while 1:
    key   = input("Enter a key: ")
    value = input("Enter a value: ")
    d1[key] = value;
    if input("exit? ") == "yes": break;
print (d1)
```

Strings

In [23]:
```
# Import the string module to get all the in-built helper methods for string
import string
```

In [ ]:

In [24]:
```
# Case change of string variables
name    = "jennifEr loves Python"

# Title case
name_t  = name.title()
print (name_t)

# Upper case
name_t  = name.upper()
print (name_t)

# Lower case
name_t  = name.lower()
print (name_t)
```

```
Jennifer Loves Python
JENNIFER LOVES PYTHON
jennifer loves python
```

In [25]:
```
# String Concatenation
fname = "jennifer"
lname = "python"
flname = fname + " " +lname

print (fname, lname)
print ("Jennifer" + " " + "Python")
# OR equivalently
print (flname.title())
```

```
jennifer python
Jennifer Python
Jennifer Python
```

In [26]:
```
# Adding WhiteSpaces
print ("Jen\nloves\npython")
print ("Jen\tloves\tpython")
print ("Jen\tloves\npython")
```

```
Jen
loves
python
Jen	loves	python
Jen	loves
python
```

In [27]:
```
# Stripping Whitespace
name1 = "    Jennifer"
name2 = "Jennifer    "
name3 = "  Jennifer  "

print (name1)
print (name2)
print (name3)
print ("----------")
print (name1.lstrip()) # lstrip() takes all extra whitespaces from left
print (name2.rstrip()) # rstrip() takes all extra whitespaces from right
print (name3.strip())  # strip() takes all extra whitespaces from both left and r
```

```
    Jennifer
Jennifer
  Jennifer
----------
Jennifer
Jennifer
Jennifer
```

In [31]:
```
# str() -> casts other data types to string
jensage = 21
#print ("Jennifer's age is " ,jensage) ## This is an error as one cannot concaten
print ("Jennifer's age is " *jensage) # This works!
```

```
Jennifer's age is Jennifer's age is Jennifer's age is Jennifer's age is Jennifer's age is Jennifer's ag
ge is Jennifer's age is Jennifer's age is Jennifer's age is Jennifer's age is Jennifer's age is Jennife
r's age is Jennifer's age is Jennifer's age is Jennifer's age is Jennifer's age is Jennifer's age is
```

In [32]:
```
# Lists, Dictionaries, Tuples and other objects can be casted to String
l_langs = ["Python", "R", "Julia"] # List
t_langs = ("Python", "R", "Julia") # Tuple
d_langs = {1: "Python", 2: "R", 3: "Julia"} # Dictionary

print ("This is a list: "       + str(l_langs))
print ("This is a tuple: "      + str(t_langs))
print ("This is a dictionary: " + str(d_langs))
```

```
This is a list: ['Python', 'R', 'Julia']
This is a tuple: ('Python', 'R', 'Julia')
This is a dictionary: {1: 'Python', 2: 'R', 3: 'Julia'}
```

In [33]:
```
help(string)
```

```
Help on module string:

NAME
    string - A collection of string constants.

DESCRIPTION
    Public module variables:

    whitespace -- a string containing all ASCII whitespace
    ascii_lowercase -- a string containing all ASCII lowercase letters
    ascii_uppercase -- a string containing all ASCII uppercase letters
    ascii_letters -- a string containing all ASCII letters
    digits -- a string containing all ASCII decimal digits
    hexdigits -- a string containing all ASCII hexadecimal digits
    octdigits -- a string containing all ASCII octal digits
    punctuation -- a string containing all ASCII punctuation characters
    printable -- a string containing all ASCII characters considered printable

CLASSES
    builtins.object
        Formatter
        Template

    class Formatter(builtins.object)
     |  Methods defined here:
     |
     |  check_unused_args(self, used_args, args, kwargs)
     |
     |  convert_field(self, value, conversion)
     |
     |  format(*args, **kwargs)
     |
     |  format_field(self, value, format_spec)
     |
     |  get_field(self, field_name, args, kwargs)
     |      # given a field_name, find the object it references.
     |      #  field_name:   the field being looked up, e.g. "0.name"
     |      #                 or "lookup[3]"
     |      #  used_args:    a set of which args have been used
     |      #  args, kwargs: as passed in to vformat
     |
     |  get_value(self, key, args, kwargs)
     |
     |  parse(self, format_string)
     |      # returns an iterable that contains tuples of the form:
     |      # (literal_text, field_name, format_spec, conversion)
     |      # literal_text can be zero length
     |      # field_name can be None, in which case there's no
     |      #  object to format and output
     |      # if field_name is not None, it is looked up, formatted
     |      #  with format_spec and conversion and then used
     |
     |  vformat(self, format_string, args, kwargs)
```

```
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class Template(builtins.object)
 |  A string class for supporting $-substitutions.
 |
 |  Methods defined here:
 |
 |  __init__(self, template)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  safe_substitute(*args, **kws)
 |
 |  substitute(*args, **kws)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  delimiter = '$'
 |
 |  flags = <RegexFlag.IGNORECASE: 2>
 |
 |  idpattern = '(?-i:[_a-zA-Z][_a-zA-Z0-9]*)'
 |
 |  pattern = re.compile('\n    \\$(?:\n      (?P<escaped>\\$)...(?P<brace...

FUNCTIONS
    capwords(s, sep=None)
        capwords(s [,sep]) -> string

        Split the argument into words using split, capitalize each
        word using capitalize, and join the capitalized words using
        join.  If the optional second argument sep is absent or None,
        runs of whitespace characters are replaced by a single space
        and leading and trailing whitespace are removed, otherwise
        sep is used to split and join the words.

DATA
    __all__ = ['ascii_letters', 'ascii_lowercase', 'ascii_uppercase', 'cap...
    ascii_letters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
    ascii_lowercase = 'abcdefghijklmnopqrstuvwxyz'
```

```
    ascii_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    digits = '0123456789'
    hexdigits = '0123456789abcdefABCDEF'
    octdigits = '01234567'
    printable = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTU...
    punctuation = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
    whitespace = ' \t\n\r\x0b\x0c'

FILE
    c:\users\devro\anaconda3\nm\envs\py36\lib\string.py
```

In [34]:

```
# Some helpful constants built-in the string module
print ("All Letters:   " + string.ascii_letters)
print ("Lowecase:      " + string.ascii_lowercase)
print ("Uppercase:     " + string.ascii_uppercase)
print ("Punctuations:  " + string.punctuation)
print ("Numbers:       " + string.digits)
print ("Hex Digits:    " + string.hexdigits)
print ("Oct Digits:    " + string.octdigits)
print ("Whitespace:    " + string.whitespace)
print ("Printable:     " + string.printable)
```

```
All Letters:   abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
Lowecase:      abcdefghijklmnopqrstuvwxyz
Uppercase:     ABCDEFGHIJKLMNOPQRSTUVWXYZ
Punctuations:  !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
Numbers:       0123456789
Hex Digits:    0123456789abcdefABCDEF
Oct Digits:    01234567
Whitespace:
 
Printable:     0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./:;<=>?@[\]^
 
```

Lists

In [35]:
```
# Simple Lists
names = ["Jennifer", "Python", "Scarlett"]
nums  = [1, 2, 3, 4, 5]
chars = ['A', 'q', 'E', 'z', 'Y']

print (names)
print (nums)
print (chars)
```

```
['Jennifer', 'Python', 'Scarlett']
[1, 2, 3, 4, 5]
['A', 'q', 'E', 'z', 'Y']
```

In [36]:
```
# Can have multiple data types in one list
rand_list = ["Jennifer", "Python", "refinneJ", 'J', '9', 9, 12.90, "Who"]
print (rand_list)
```

```
['Jennifer', 'Python', 'refinneJ', 'J', '9', 9, 12.9, 'Who']
```

In [37]:
```
# Accessing elements in a list
# 0-indexed
print (names[2])
print (rand_list[3])
print (names[0] + " " + rand_list[2].title())
```

```
Scarlett
J
Jennifer Refinnej
```

In [38]:
```
# Negetive indexes: Access elements from the end of the list without knowing the
print (rand_list[-1]) # Returns the last element of the list [1st from the end]
print (rand_list[-2]) # Returns the 2nd last element
# and so on..
```

```
Who
12.9
```

In [39]:
```
# Now here's a question.
print (rand_list[-1] + " is " + names[2] + "?")
print ("A) " + rand_list[0] + "'s sister\tB) " + names[0] + "'s Friend\nC) Not Re
```

```
Who is Scarlett?
A) Jennifer's sister    B) Jennifer's Friend
C) Not Related to Jennifer      D) Nice question but I don't know
```

In [40]:
```
# Modifying elements in a list
str_list = ["Scarlett", "is", "a", "nice", 'girl', '!']
#mutable = mutation
print (str_list)
str_list[0] = "Jennifer"
print (type(str_list[0]))
```

```
['Scarlett', 'is', 'a', 'nice', 'girl', '!']
<class 'str'>
```

In [41]:
```
# Adding elements to a list
# Use append() to add elements to the end of the list
str_list.append('She is 21.')
print (str_list)
```

```
['Jennifer', 'is', 'a', 'nice', 'girl', '!', 'She is 21.']
```

In [42]:
```
# So, you can build lists like this
my_list = []
my_list.append ("myname")
my_list.append ("myage")
my_list.append ("myaddress")
my_list.append ("myphn")
my_list.append ("is")
my_list.append (1234567890)
print (my_list)
```

```
['myname', 'myage', 'myaddress', 'myphn', 'is', 1234567890]
```

In [43]:
```
# Insert elements at specific positions of the list
# insert(index, element)
my_list.insert (0, "Mr/Miss/Mrs")
print (my_list)

my_list.insert(4, "mybday")
print (my_list)
```

```
['Mr/Miss/Mrs', 'myname', 'myage', 'myaddress', 'myphn', 'is', 1234567890]
['Mr/Miss/Mrs', 'myname', 'myage', 'myaddress', 'mybday', 'myphn', 'is', 1234567890]
```

In [44]:
```
# Using '-1' to insert at the end doesn't work and inserts element at the 2nd las
my_list = ['A', 'B', 'C', 'D']
my_list.insert (-1, 'E')
print (my_list)
my_list.insert (-1, 'E')
```

```
['A', 'B', 'C', 'E', 'D']
```

In [45]:
```
# Using '-2' inserts at 3rd last position
# In general, use '-n' to insert at 'n+1'th position from end.
my_list = ['A', 'B', 'C', 'D']
my_list.insert (-2, 'E')
print (my_list)
```

```
['A', 'B', 'E', 'C', 'D']
```

In [46]:
```
# Insert elements at the end
l1 = ['A', 'B', 'C', 'D']
l2 = ['A', 'B', 'C', 'D']

l1.append('E')
l2.insert(len(my_list), 'E')
print (l1)
print (l2)
```

```
['A', 'B', 'C', 'D', 'E']
['A', 'B', 'C', 'D', 'E']
```

In [47]:
```
# Length of the list
l1 = ['A', 'B', 'C', 'D', 'E']
print (len(l1))
```

```
5
```

In [48]:
```
# # Removing elements from list
# del can remove any element from list as long as you know its index
l1 = ['A', 'B', 'C', 'D', 'E']
print (l1)

del l1[0]
print (l1)

del l1[-1]
print (l1)
```

```
['A', 'B', 'C', 'D', 'E']
['B', 'C', 'D', 'E']
['B', 'C', 'D']
```

In [51]:
```
# pop() can remove the last element from list when used without any arguments
l1 = ['A', 'B', 'C', 'D', 'E']
# pop() returns the last element, so c would store the popped element
c = l1.pop()
d = l1.pop()
print (l1)
print (c)
```

```
['A', 'B', 'C']
E
```

In [52]:
```
# pop(n) -> Removes the element at index 'n' and returns it
l1 = ['A', 'B', 'C', 'D', 'E']

# Removes the element at 0 position and returns it
c = l1.pop(0)
print (l1)
print (c)

# Works as expected with negetive indexes
c = l1.pop(-1)
print (l1)
print (c)
```

```
['B', 'C', 'D', 'E']
A
['B', 'C', 'D']
E
```

In [53]:
```
# Removing an item by value
# remove() only removes the first occurence of the value that is specified.
q1 = ["Seriously, ", "what", "happened", "to", "Jennifer", "and", "Jennifer", "?"
print (q1)

q1.remove ("Jennifer")
print (q1)

n1 = "and"
q1.remove(n1)
print (q1)
```

```
['Seriously, ', 'what', 'happened', 'to', 'Jennifer', 'and', 'Jennifer', '?']
['Seriously, ', 'what', 'happened', 'to', 'and', 'Jennifer', '?']
['Seriously, ', 'what', 'happened', 'to', 'Jennifer', '?']
```

In [54]:
```
# Sorting a list
# sort() -> sorts list in increasing or decreasing order, *permantantly*
# Sorts in alphabetical order
l1 = ['E', 'D', 'C', 'B', 'A']
l1.sort()
print (l1)

# Sorts in increasing order
l2 = [2, 200, 16, 4, 1, 0, 9.45, 45.67, 90, 12.01, 12.02]
l2.sort()
print (l2)
```

```
['A', 'B', 'C', 'D', 'E']
[0, 1, 2, 4, 9.45, 12.01, 12.02, 16, 45.67, 90, 200]
```

In [55]:
```
help(list.sort)
```

```
Help on method_descriptor:

sort(...)
    L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE*
```

In [56]:
```
# Reverse sorts alphabetical order
l1 = ['E', 'D', 'C', 'B', 'A']
l1.sort(reverse=True)
print (l1)

# Sorts in decreasing order
l2 = [2, 200, 16, 4, 1, 0, 9.45, 45.67, 90, 12.01, 12.02]
l2.sort(reverse=True)
print (l2)
```

```
['E', 'D', 'C', 'B', 'A']
[200, 90, 45.67, 16, 12.02, 12.01, 9.45, 4, 2, 1, 0]
```

In [57]:
```
# sorted() -> Sorts list in increasing or decreasing order, *temporarily*
# Sorts in increasing order
l2 = [2, 200, 16, 4, 1, 0, 9.45, 45.67, 90, 12.01, 12.02]
print (l2)
print (sorted(l2))
print (l2)
```

```
[2, 200, 16, 4, 1, 0, 9.45, 45.67, 90, 12.01, 12.02]
[0, 1, 2, 4, 9.45, 12.01, 12.02, 16, 45.67, 90, 200]
[2, 200, 16, 4, 1, 0, 9.45, 45.67, 90, 12.01, 12.02]
```

In [58]:
```
# Sorts in decreasing order
l2 = [2, 200, 16, 4, 1, 0, 9.45, 45.67, 90, 12.01, 12.02]
print (l2)
print (sorted(l2, reverse=True))
print (l2)
```

```
[2, 200, 16, 4, 1, 0, 9.45, 45.67, 90, 12.01, 12.02]
[200, 90, 45.67, 16, 12.02, 12.01, 9.45, 4, 2, 1, 0]
[2, 200, 16, 4, 1, 0, 9.45, 45.67, 90, 12.01, 12.02]
```

In [59]:
```
# Reverse list
l1 = ['E', 'D', 'C', 'B', 'A']
l1.reverse()
print (l1)
```

```
['A', 'B', 'C', 'D', 'E']
```

In [60]:
```
# Looping Through a list using for
l1 = ["Scarlett", "is", "now", "back", "from", "her first", "Python", "lesson."]

# Do notice the indentations
for xyz in l1:
    print (xyz, end = " ")
```

```
Scarlett is now back from her first Python lesson.
```

In [61]:
```
# Looping through a list using while
l1 = ["Scarlett", "is", "in", "love", "with", "Python"]
i = 0
while i is not len(l1):
    print (l1[i])
    i += 1
```

```
Scarlett
is
in
love
with
Python
```

In [63]:
```
# Numerical lists
# Note: range(n, m) will loop over numbers from n to m-1
l1 = ['A', 'B', 'C', 'D', 'E']
print ("Guess how much Scarlett scored in her first lesson out of 5:")
for val in range(5):
    print (l1[val] + ") " + str(val+1))
```

```
Guess how much Scarlett scored in her first lesson out of 5:
A) 1
B) 2
C) 3
D) 4
E) 5
```

In [ ]:

In [64]:
```
# Using range() to make a list of numbers
num_list = list(range(1, 6))
print (num_list)
```

```
[1, 2, 3, 4, 5]
```

In [65]:
```
# Use range() to skip values at intervals
# range (num_to_start_from, num_to_end_at+1, interval)
l1 = list(range(10, 51, 5))
print (l1)
```

```
[10, 15, 20, 25, 30, 35, 40, 45, 50]
```

In [66]:
```
# Operations with list of numbers with -> min() max() sum()
l1 = [2, 3, 4, 45, 1, 5, 6, 3, 1, 23, 14]

print ("Sum: " + str(sum(l1)))
print ("Max: " + str(max(l1)))
print ("Min: " + str(min(l1)))
```

```
Sum: 107
Max: 45
Min: 1
```

In [67]:
```
# List Comprehensions
# Simple
l1 = [x for x in range(1,100)]
l2 = [i+1 for i in range(20, 30, 1)]
l3 = [[i, i**2] for i in range(2, 12, 3)]
print (l1)
print (l2)
print (l3)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
7, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
[21, 22, 23, 24, 25, 26, 27, 28, 29, 30]
[[2, 4], [5, 25], [8, 64], [11, 121]]
```

In [68]:
```
# A few more list comprehension examples
equi_list_1 = [[x, y, z] for x in range(1, 3) for y in range(3, 6) for z in range
print (equi_list_1)
```

```
[[1, 3, 6], [1, 3, 7], [1, 3, 8], [1, 4, 6], [1, 4, 7], [1, 4, 8], [1, 5, 6], [1, 5, 7], [1, 5, 8], [2,
4, 6], [2, 4, 7], [2, 4, 8], [2, 5, 6], [2, 5, 7], [2, 5, 8]]
```

In [69]:
```
# The above list comprehension is equivalent of the following code
equi_list_2 = []
for x in range(1, 3):
    for y in range(3, 6):
        for z in range(6, 9):
            equi_list_2.append([x, y, z])
print (equi_list_2)
```

```
[[1, 3, 6], [1, 3, 7], [1, 3, 8], [1, 4, 6], [1, 4, 7], [1, 4, 8], [1, 5, 6], [1, 5, 7], [1, 5, 8], [2,
4, 6], [2, 4, 7], [2, 4, 8], [2, 5, 6], [2, 5, 7], [2, 5, 8]]
```

In [70]:
```
# Proof of equivalence  (Do execute the above two blocks of code before running t
print (equi_list_1 == equi_list_2)
```

```
True
```

In [71]:
```
# List Comprehension with conditionals
l1 = [x if x%5==0 else "blank" for x in range(20)]
print (l1)
```

```
[0, 'blank', 'blank', 'blank', 'blank', 5, 'blank', 'blank', 'blank', 'blank', 10, 'blank', 'blank', 'b
k', 'blank', 'blank']
```

In [72]:
```
# One more list comprehension with conditionals
l1 = ["Jennifer", "met", "Scarlett", "in", "Python", "lessons", "they", "take."]
l2 = [[str(x) + ") " + y] for x in range(len(l1)) for y in l1 if l1[x] == y]
print (l2)
```

```
[['0) Jennifer'], ['1) met'], ['2) Scarlett'], ['3) in'], ['4) Python'], ['5) lessons'], ['6) they'], [
```

In [73]:
```
# Slicing a list
l1 = ["Jennifer", "is", "now", "friends", "with", "Scarlett"]

# [start_index : end_index+1]
print("[2:5] --> " + str(l1[2:5]))
print("[:4]  --> " + str(l1[:4]))  # everthing before 4th index [excluding the 4t
print("[2:]  --> " + str(l1[2:]))  # everything from 2nd index [including the 2nd
print("[:]   --> " + str(l1[:]))   # every element in the list
```

```
[2:5] --> ['now', 'friends', 'with']
[:4]  --> ['Jennifer', 'is', 'now', 'friends']
[2:]  --> ['now', 'friends', 'with', 'Scarlett']
[:]   --> ['Jennifer', 'is', 'now', 'friends', 'with', 'Scarlett']
```

In [74]:
```
# Some more slicing
l1 = ["Jennifer", "and", "Scarlett", "now", "Pythonistas", "!"]

print ("[-2:]   --> " + str(l1[-2:]))
print ("[:-3]   --> " + str(l1[:-3]))
print ("[-5:-2] --> " + str(l1[-5:-2]))
print ("[-4:-6] --> " + str(l1[-4:-6]))
```

```
[-2:]   --> ['Pythonistas', '!']
[:-3]   --> ['Jennifer', 'and', 'Scarlett']
[-5:-2] --> ['and', 'Scarlett', 'now']
[-4:-6] --> []
```

In [75]:
```
# Looping through a slice
l1 = ["Pythonistas", "rock", "!!!", "XD"]
for idx, w in enumerate(l1[:]):
    print (w.upper())
    print(idx)
```

```
PYTHONISTAS
0
ROCK
1
!!!
2
XD
3
```

In [76]:
```
# Copying a list
l1 = ["We", "should", "use", "[:]", "to", "copy", "the", "whole", "list"]
l2 = l1[:]
print(l2)
```

```
['We', 'should', 'use', '[:]', 'to', 'copy', 'the', 'whole', 'list']
```

In [77]:
```
# Proof that the above two lists are different
l2.append(". Using [:] ensures the two lists are different")

print (l1)
print (l2)
```

```
['We', 'should', 'use', '[:]', 'to', 'copy', 'the', 'whole', 'list']
['We', 'should', 'use', '[:]', 'to', 'copy', 'the', 'whole', 'list', '. Using [:] ensures the two lists
```

In [78]:
```
# What happens if we directly assign one list to the other instead of using slice
l1 = ["Jennifer", "now", "wonders", "what", "happens", "if", "we", "directly", "a
l2 = l1
l2.append("Both variables point to the same list")

print (l1)
print (l2)
```

```
['Jennifer', 'now', 'wonders', 'what', 'happens', 'if', 'we', 'directly', 'assign.', 'Both variables po
['Jennifer', 'now', 'wonders', 'what', 'happens', 'if', 'we', 'directly', 'assign.', 'Both variables po
```

Tuples

In [79]:
```
# Simple Tuples
# Tuples are list, the elements stored in which cannot be changed.
t1 = (23, 45)
print (t1)
print (t1[1])
```

```
(23, 45)
45
```

In [80]:
```
# Can have more than two elements
t1 = (1, 2, 3, 4, 5, 6, 7)
print (t1)
print (t1[4])
```

```
(1, 2, 3, 4, 5, 6, 7)
5
```

In [81]:
```
# Cannot change the elements stored as compared to lists
l1 = [1, 2, 3, 4, 5, 6, 7]
t1 = (1, 2, 3, 4, 5, 6, 7)

# Can change elements in list
l1[4] = 1
# Cannot change elements in tuple (Uncomment it to see the error)
t1[4] = 1
#immutable
print (l1)
print (t1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-81-2a5f2968086c> in <module>
      6 l1[4] = 1
      7 # Cannot change elements in tuple (Uncomment it to see the error)
----> 8 t1[4] = 1
      9 #immutable
     10 print (l1)

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

In [82]:
```
# Looping over a tuple
t1 = (34, 12, 56, 78, 89)
for t in t1:
    print (t)
```

```
34
12
56
78
89
```

In [83]:
```
# Writing over a tuple
# One cannot change the elements stored in the tuple.
# But you can assign a new tuple to the variable that stores the tuple you wanna
t1 = (23, 12, 45, 78)
print (t1)

t1 = (45, 67) # t1 changed to new value
print (t1)
```

```
(23, 12, 45, 78)
(45, 67)
```

Dictionaries

In [84]:
```
# Simple Dictionary
# Dictionary allows to have key:value pairs
d1 = {"Jennifer":8, 'A':65, 66:'B', 9.45:"Decimals"}
Roll = {"ABC":12, "POIUY":13}
print(Roll["ABC"])


print (d1["Jennifer"])
print (d1['A'])
print (d1[66])
print (d1[9.45])
```

```
12
8
65
B
Decimals
```

In [85]:
```
# Adding new key:value pairs
d1 = {"Jennifer":8, 'A':65, 66:'B', 9.45:"Decimals"}

d1["Scarlett"] = 8
d1[7.56]       = "Is a decimal!"
d1['Q']        = 17

print (d1["Scarlett"])
print (d1[7.56])
print (d1['Q'])



print (d1)
```

```
8
Is a decimal!
17
{'Jennifer': 8, 'A': 65, 66: 'B', 9.45: 'Decimals', 'Scarlett': 8, 7.56: 'Is a decimal!', 'Q': 17}
```

In [ ]:
```

```

In [ ]:
```
# Declaring an empty dictionary
d1 = {}

# Add new values
d1["Jennifer"] = "Python"
d1["Scarlett"] = "Python"
d1[45]         = 56

print (d1)
```

In [ ]:
```
# Modifying values in a dictionary
d1 = {"Python":"Is a language", "Jennifer":"Feels like a supergirl with Python"}

d1["Python"] = "Is Love"
d1["Jennifer"] = 8
print (d1)
```

In [86]:
```
# Removing Key:Value pairs
d1 = {"Key":"Value", "Jennifer":"Scarlett", "Scarlett":"Jennifer"}

del d1["Key"]
print (d1)
```

```
{'Jennifer': 'Scarlett', 'Scarlett': 'Jennifer'}
```

In [ ]:
```
# Storing a dictionary inside a dictionary
d1 = {'A':65, 'B':66, 'C':67, 'D': {
    "Breaking": "Dict into dicts",
    "Inception": "All over",
    '!': "XD"},
    'E':69,
    "What happened with D?": "It became a 'D'ictionary! XD",
    66:66}

print (d1['D']["Inception"])
print (d1["What happened with D?"])
print ('\n')
print (d1)
```

In [ ]:
```
# Looping through a dictionary
x,y = 1,2


for key, value in d1.items():
    print("Key: "    + str(key))
    print ("Value: " + str(value))
    print('\n')
```

In [ ]:
```
# Looping through the keys in dictionary
for key in d1.keys():
    print("Key: "    + str(key))
    print ("Value: " + str(d1[key]))
    print('\n')
```

In [ ]:
```
# Note: Default behaviour is to loop through keys if not specified d1.keys()
for k in d1:
    print("Key: "    + str(k))
```

In [ ]:
```
# Check if a particular key is  present
if 'F' not in d1.keys():
    print ("What happened to F?")
```

In [ ]:
```
# Looping through dictonary keys in sorted order (increasing)
for key in sorted(d1.keys()):
    print ("Key :" + key + '\t' + "Value :" + str(d1[key]) + '\n')
```

In [ ]:
```
# Looping through dictonary keys in sorted order (decreasing)
for key in sorted(d1.keys(), reverse=True):
    print ("Key :" + key + '\t' + "Value :" + str(d1[key]) + '\n')
```

In [ ]:
```
# Looping through values in dictionary (with repeats)
# Note: If two or more keys have the same value then this method will print all o
for value in d1.values():
    print (str(value).title())
```

In [ ]:
```
# List in Dictionary
d1 = {"l1":['A', 'B', 'C', 'D'],
      "l2":['E', 'F', 'G', 'H'],
       45 : "qwerty",
      '$' : "$Dollar$"}

# Accessing the elements in list
print (d1["l1"][2])
print (d1["l2"][-1])
```

In [ ]:
```
# Looping over just the lists in dictionary
for k in d1.keys():
    if type(d1[k]) == list:
        print ("List :" + k)
        for val in d1[k]:
            print (val)
        print ('\n')
```

Functions

In [ ]:
```
# Simple function
# Function definition
def a_func():
    print ("A Message from the other world!")

# Function Call
a_func()
a_func()
```

In [ ]:
```
# Passing arguments to functions
def add_two(num):
    '''This function adds 2 to a given number'''
    return (int(num)+2)

x = add_two (3)
y = add_two ("45") # This will work as we are casting the passed parameter to Int

print(x,y)
```

In [ ]:
```
# Multiple arguments
def add_sub(add_two_to_this, sub_two_from_this):
    print ("Added 2 to : " + str(add_two_to_this))
    print ("Answer: " + str(int(add_two_to_this)+2))

    print ("Subtracted 3 from : " + str(sub_two_from_this))
    print ("Answer: " + str(int(sub_two_from_this)-2))

add_sub (45, "67")
add_sub ("-156745", 12131)
```

In [ ]:
```
# Ordering of passed argument matters
def coding_in(name, language):
    print (str(name) + " is coding in " + str(language))

coding_in("Jennifer", "Python")
# If you change the order, results are unexpected
coding_in("Python", "Jennifer")
```

In [ ]:
```
# Keyword Arguments
# Pass the argumnets as key:value pairs, so even if unordered it won't produce un
def coding_in(name, language):
    print (str(name) + " is coding in " + str(language))

coding_in(name="Jennifer", language="Python")
# If you change the order, results are same
coding_in(language="Python", name="Jennifer")
```

In [ ]:
```
# Default Values for parameters
# Note: If you do not pass arguments required by the function and that argument a
#       then python will throw an error
def coding_in(name, language="Python"):
    print (str(name) + " is coding in " + str(language))

coding_in("Jennifer") # Since 2nd argument is not passed, it takes on the default
coding_in("Jennifer", "R") # Since 2nd argument is passed, it takes on the passea
```

In [ ]:
```
# Note: You cannot have parameters with no default value after parameters having
# Following is an error
def coding_in(name, language1="Python", language2):
    print (str(name) + " is coding in " + str(language1) + " and " + str(language

coding_in("Jennifer", "Python", "R")
```

In [ ]:
```
# Easy fix to above error is to declare all non-default parameters,
# and then start declaring the default parameters
# Note: Default parameters can be used to make an argument optional
def coding_in(name, language2, language1="Python"):
    print (str(name) + " is coding in " + str(language1) + " and " + str(language

coding_in("Jennifer", "Python", "R")
coding_in("Jennifer", "R")
```

In [ ]:
```
# All parameters can be default
def coding_in(name="Jennifer", language1="Python", language2="R"):
    print (str(name) + " is coding in " + str(language1) + " and " + str(language

coding_in()
```

In [ ]:
```
# return
def pow_4(num):
    return num**4

v1 = pow_4(34) # v1 now stores 34^4
v2 = pow_4(23) # v2 now stores 23^4

print ("34^4 = " + str(v1))
print ("23^4 = " + str(v2))
```

In [ ]:
```
# can return any data type or object from function
def make_a_coder(name, age, language):
    d1 = {'name': name,
          'age' : age,
          'language' : language}
    return d1

print(make_a_coder("Jennifer", "21", "Python"))
```

In [ ]:
```
# can pass any data type or object to function
def make_many_coders(list_of_coders):
    print ("Name    \tLanguage  \tAge")
    print ("========\t==========\t===")
    for coder,details in list_of_coders.items():
        print (str(coder) + "\t" + str(details[0]) + "\t\t" + str(details[-1]))

    return str(len(list_of_coders))

d1 = {"Jennifer": ["Python", 21],
      "Scarlett": ["R", 21]}
print ('\n' + make_many_coders(d1) + " coders found!")
```

In [ ]:
```
# If a list passed to a function is changed inside the function then the change i
def make_language_list(language_list, new_language):
    language_list.append(new_language)

lang_list = ["Python", "R"]
print (lang_list)

make_language_list(lang_list, "Julia")
print (lang_list)
```

In [ ]:
```
# Preventing a function from modifying a list
def make_language_list(language_list, new_language):
    language_list.append(new_language)

lang_list = ["Python", "R"]
print (lang_list)

make_language_list(lang_list[:], "Julia")
print (lang_list)
```

In [ ]:
```
# Passing Arbitrary number of arguments
# The '*' tells Python to make a tuple of whatever number of arguments it receive
def languages(*many_languages):
    print (many_languages)

languages ("Python")
languages ("Python", "R", "Julia", "Ruby", "Go")
```

In [ ]:
```
# Passing Arbitrary number of arguments with a normal argument
# Note: The parameter that accepts arbitrary number of arguments needs to be plac
def knows_languages(name, *languages):
    print (str(name) +  " knows: " + str(languages))

knows_languages("Jennifer", "Python")
knows_languages("Jennifer", "Python", "R", "Julia", "Ruby")
```

In [ ]:
```
# Note: You cannot have two or more parameters that accepts arbitrary number of a
# Hence, following is an error!
def knows_languages_and_modules(name, *languages, *modules):
    print (str(name) +  " knows: " + str(languages))

knows_languages("Jennifer", "Python", "PyGtk")
knows_languages("Jennifer", "Python", "R", "Julia", "Ruby", "PyGtk", "PyGame", "a
```

In [ ]:
```
# Using Arbitrary Keyword Argument
# Note: '**' tells Python to create a dictionary of the extra arguments passed af
def make_a_coder(name, **details):
    coder = {}
    coder["name"] = name;
    for key, value in details.items():
        coder[key] = value
    return coder

print (make_a_coder("Jennifer", location="California", age="21", language=("Pytho

# We can do this since we are using keyword arguments
print (make_a_coder(location="California", age="21", language=("Python", "R"), na
```

Files

In [ ]:
```
# Opening a file
'''
Make sure you have a file in your directory before running this code.
If you have a file but still getting errors, then make sure the path is correct.
'''
# Be sure to close a file after use
file = open ("./others/test.txt")
print (file.read())
file.close()
```

In [ ]:
```
# Modes to open a file
file = open ("./others/test.txt", 'r')  # Opens in read mode
file = open ("./others/test.txt", 'w')  # Opens in write mode
file = open ("./others/test.txt", 'a')  # Opens in append mode
file = open ("./others/test.txt", 'r+') # Allows to read and write

file.close()
```

In [ ]:
```
# Reading the entire file at once
# 'with' keeps the file open as long as its needed then closes it
with open ("./others/test.txt") as file:
    content = file.read()
    print (content)
```

In [ ]:
```
# Reading the file line by line
with open ("./others/test.txt") as file:
    i = 0
    for line in file:
        i += 1
        print ("Line " + str(i) + " : " + str(line))
```

In [ ]:
```
# Writing to an empty file
# When opening a file in 'w' mode, if the files contains something it will get er
# To retain the contents of the file open in 'a' mode, namely append mode
with open("./others/test.txt", 'w') as file:
    file.write ("Python is Love!")
    file.write ("This is a second line.")
    file.write ("Python doesn't add new lines on its own.")
    file.write ("\nSo lets provide some new lines.\n")
    file.write ("\nBy the way, you just wrote multiple lines to a file :)")
    file.write ("\nAnd just remember 'with' closes the opened file.")
    file.write (" So you do not have to close it yourself. When opened a file wit
```

In [ ]:
```
# Appending to a file
with open("./others/test.txt", 'a') as file:
    file.write ("\nWell now the text in the opened file is retained.")
    file.write ("\nAnd you are appending text to already present text.")
    file.write ("\nCool!")
```

Numpy

In [ ]:
```
import numpy as np
```

In [91]:
```
a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"



b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)
print('b[0,0] : ',b[0, 0],'b[0, 1] : ' ,b[0, 1],' b[1, 0] : ',b[1, 0])
```

```
<class 'numpy.ndarray'>
(3,)
1 2 3
[5 2 3]
(2, 3)
b[0,0] :  1 b[0, 1] :  2  b[1, 0] :  4
```

In [92]:
```
a = np.zeros((2,2))   # Create an array of all zeros
print(a)
```

```
[[0. 0.]
 [0. 0.]]
```

In [93]:
```
b = np.ones((1,2))    # Create an array of all ones
print(b)
```

```
[[1. 1.]]
```

In [94]:
```
c = np.full((2,2), 7)  # Create a constant array
print(c)
```

```
[[7 7]
 [7 7]]
```

In [95]:
```
d = np.eye(2)         # Create a 2x2 identity matrix
print(d)
```

```
[[1. 0.]
 [0. 1.]]
```

In [98]:
```
e = np.rand((2,2))  # Create an array filled with random values
print(e)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-98-6770bc02e8dc> in <module>
----> 1 e = np.randn((2,2))  # Create an array filled with random values
      2 print(e)

AttributeError: module 'numpy' has no attribute 'randn'
```

In [ ]:
```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
```

In [ ]:
```
b = a[:2, 1:3]
print(b)
```

In [99]:
```
# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print('before a[0,1] : ',a[0,1])
b[0, 0] = 77
print('after a[0,1] : ',a[0, 1])
```

```
before a[0,1] :  0.0
after a[0,1] :  0.0
```

In [100]:
```
row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)
print(row_r2, row_r2.shape)
```

```
[0. 0.] (2,)
[[0. 0.]] (1, 2)
```

In [101]:
```
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)
print(col_r2, col_r2.shape)
```

```
[0. 0.] (2,)
[[0.]
 [0.]] (2, 1)
```

In [102]:
```
a = np.array([[1,2], [3, 4], [5, 6]])
print(a[[0, 1, 2], [0, 1, 0]])
```

```
[1 4 5]
```

In [103]:
```
# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))
```

```
[1 4 5]
```

In [104]:
```
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
print(a)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

In [105]:
```
b = np.array([0, 2, 0, 1])
```

In [106]:
```
# Select one element from each row of a using the indices in b
print(a[np.arange(4), b])
```

```
[ 1  6  7 11]
```

In [107]:
```
a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)

print(bool_idx)

# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print(a[bool_idx])
```

```
[[False False]
 [ True  True]
 [ True  True]]
[3 4 5 6]
```

In [108]:
```
# We can do all of the above in a single concise statement:
print(a[a > 2])
```

```
[3 4 5 6]
```

In [109]:
```
x = np.array([1, 2])   # Let numpy choose the datatype
print(x.dtype)

x = np.array([1.0, 2.0])   # Let numpy choose the datatype
print(x.dtype)

x = np.array([1, 2], dtype=np.int64)   # Force a particular datatype
print(x.dtype)
```

```
int32
float64
int64
```

In [110]:

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

print('sum : ' ,x + y)
print('sum : ' ,np.add(x, y))

print('subtraction : ' ,x - y)
print('subtraction : ',np.subtract(x, y))


print('multiplication : ' ,x * y)
print('multiplication : ' ,np.multiply(x, y))

print('division : ' ,x / y)
print('division : ' ,np.divide(x, y))


print('square root : ' ,np.sqrt(x))
```

```
sum :  [[ 6.  8.]
 [10. 12.]]
sum :  [[ 6.  8.]
 [10. 12.]]
subtraction :  [[-4. -4.]
 [-4. -4.]]
subtraction :  [[-4. -4.]
 [-4. -4.]]
multiplication :  [[ 5. 12.]
 [21. 32.]]
multiplication :  [[ 5. 12.]
 [21. 32.]]
division :  [[0.2        0.33333333]
 [0.42857143 0.5       ]]
division :  [[0.2        0.33333333]
 [0.42857143 0.5       ]]
square root :  [[1.         1.41421356]
 [1.73205081 2.        ]]
```

In [111]:
```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))
```

```
219
219
```

In [112]:
```
# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))
```

```
[29 67]
[29 67]
```

In [113]:
```
# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

```
[[19 22]
 [43 50]]
[[19 22]
 [43 50]]
```

In [116]:
```
x = np.array([[1,2],[3,4]])
print(x)
print(np.sum(x,axis=0))  # Compute sum of all elements; prints "10"
```

```
[[1 2]
 [3 4]]
[4 6]
```

In [117]:
```
print(np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"
```

```
[4 6]
[3 7]
```

In [118]:
```
x = np.array([[1,2], [3,4]])
print(x)
print(x.T)

# Note that taking the transpose of a rank 1 array does nothing:
v = np.array([1,2,3])
print(v)
print(v.T)
```

```
[[1 2]
 [3 4]]
[[1 3]
 [2 4]]
[1 2 3]
[1 2 3]
```

In [119]:
```
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)   # Create an empty matrix with the same shape as x
# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v


print(y)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [120]:
```
print(x+v)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [121]:
```
import numpy as np

# Compute outer product of vectors
v = np.array([1,2,3])  # v has shape (3,)
w = np.array([4,5])    # w has shape (2,)
# To compute an outer product, we first reshape v to be a column
# vector of shape (3, 1); we can then broadcast it against w to yield
# an output of shape (3, 2), which is the outer product of v and w:
# [[ 4  5]
#  [ 8 10]
#  [12 15]]
print(np.reshape(v, (3, 1)) * w)

x = np.array([[1,2,3], [4,5,6]])



# Add a vector to each column of a matrix
# x has shape (2, 3) and w has shape (2,).
# If we transpose x then it has shape (3, 2) and can be broadcast
# against w to yield a result of shape (3, 2); transposing this result
# yields the final result of shape (2, 3) which is the matrix x with
# the vector w added to each column. Gives the following matrix:

print((x.T + w).T)
```

```
[[ 4  5]
 [ 8 10]
 [12 15]]
[[ 5  6  7]
 [ 9 10 11]]
```

In [122]:
```
# Another solution is to reshape w to be a column vector of shape (2, 1);
# we can then broadcast it directly against x to produce the same
# output.
print(x + np.reshape(w, (2, 1)))
```

```
[[ 5  6  7]
 [ 9 10 11]]
```

Pandas

In [123]:
```
import pandas as pd
import numpy as np
```

In [124]:
```
df=pd.DataFrame((np.arange(30).reshape(5,6)),index=['row'+ i for i in ['1','2','3
```

In [125]:
```
df
```

	column1	column2	column3	column4	column5	column6
row1	0	1	2	3	4	5
row2	6	7	8	9	10	11
row3	12	13	14	15	16	17
row4	18	19	20	21	22	23
row5	24	25	26	27	28	29

In [127]:
```
df.to_csv('example.csv')
```

In [128]:
```
df = pd.read_csv('example.csv')
df
```

	Unnamed: 0	column1	column2	column3	column4	column5	column6
0	row1	0	1	2	3	4	5
1	row2	6	7	8	9	10	11
2	row3	12	13	14	15	16	17
3	row4	18	19	20	21	22	23
4	row5	24	25	26	27	28	29

In [129]:
```
df = pd.read_csv('example.csv',usecols = ['column1','column2'])
df
```

	column1	column2
0	0	1
1	6	7
2	12	13
3	18	19
4	24	25

In [130]:
```
df = pd.read_csv('example.csv',index_col=0)
df
```

	column1	column2	column3	column4	column5	column6
row1	0	1	2	3	4	5
row2	6	7	8	9	10	11
row3	12	13	14	15	16	17
row4	18	19	20	21	22	23
row5	24	25	26	27	28	29

In [131]:
```
df.describe()
```

	column1	column2	column3	column4	column5	column6
count	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000
mean	12.000000	13.000000	14.000000	15.000000	16.000000	17.000000
std	9.486833	9.486833	9.486833	9.486833	9.486833	9.486833
min	0.000000	1.000000	2.000000	3.000000	4.000000	5.000000
25%	6.000000	7.000000	8.000000	9.000000	10.000000	11.000000
50%	12.000000	13.000000	14.000000	15.000000	16.000000	17.000000
75%	18.000000	19.000000	20.000000	21.000000	22.000000	23.000000
max	24.000000	25.000000	26.000000	27.000000	28.000000	29.000000

In [132]:
```
## accessing element
## .loc
## .iloc
df.loc['row2']
```

```
column1     6
column2     7
column3     8
column4     9
column5    10
column6    11
Name: row2, dtype: int64
```

In [133]:
```
df.iloc[1,:]
```

```
column1     6
column2     7
column3     8
column4     9
column5    10
column6    11
Name: row2, dtype: int64
```

In [134]:
```
df['column1']
```

```
row1     0
row2     6
row3    12
row4    18
row5    24
Name: column1, dtype: int64
```

In [135]:
```
df['column1'].value_counts()
```

```
6     1
12    1
24    1
18    1
0     1
Name: column1, dtype: int64
```

In [136]:
```
df['column1'].unique()
```

```
array([ 0,  6, 12, 18, 24], dtype=int64)
```

In [137]:
```
df[df['column1']==6]
```

	column1	column2	column3	column4	column5	column6
row2	6	7	8	9	10	11

In [138]:
```
df.iloc[1,2] = np.nan
```

In [139]:
```
df
```

	column1	column2	column3	column4	column5	column6
row1	0	1	2.0	3	4	5
row2	6	7	NaN	9	10	11
row3	12	13	14.0	15	16	17
row4	18	19	20.0	21	22	23
row5	24	25	26.0	27	28	29

In [140]:
```
df.isnull()
```

	column1	column2	column3	column4	column5	column6
row1	False	False	False	False	False	False
row2	False	False	True	False	False	False
row3	False	False	False	False	False	False
row4	False	False	False	False	False	False
row5	False	False	False	False	False	False

In [141]:
```
df['column3'].value_counts()
```

```
26.0    1
20.0    1
14.0    1
2.0     1
Name: column3, dtype: int64
```

In [142]:
```
df['column3'].unique()
```

```
array([ 2., nan, 14., 20., 26.])
```

In [143]:
```
df['column3'].isnull().sum()
```

```
1
```

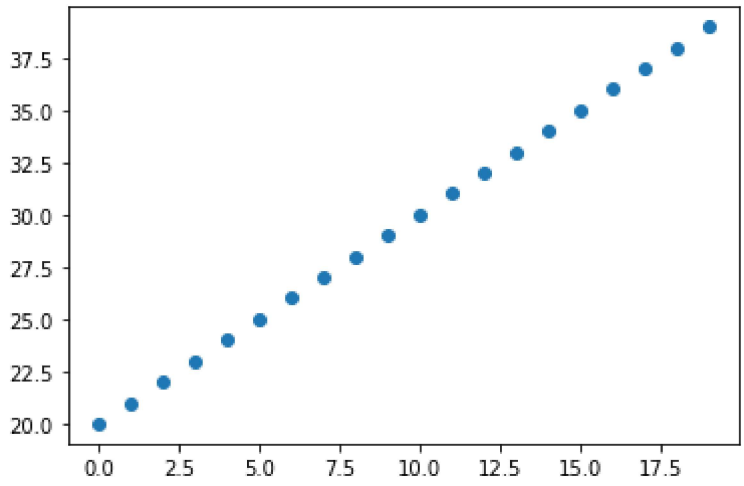
matplotlib

In [144]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [145]:
```
x=np.arange(20)
y=np.arange(20,40)
```

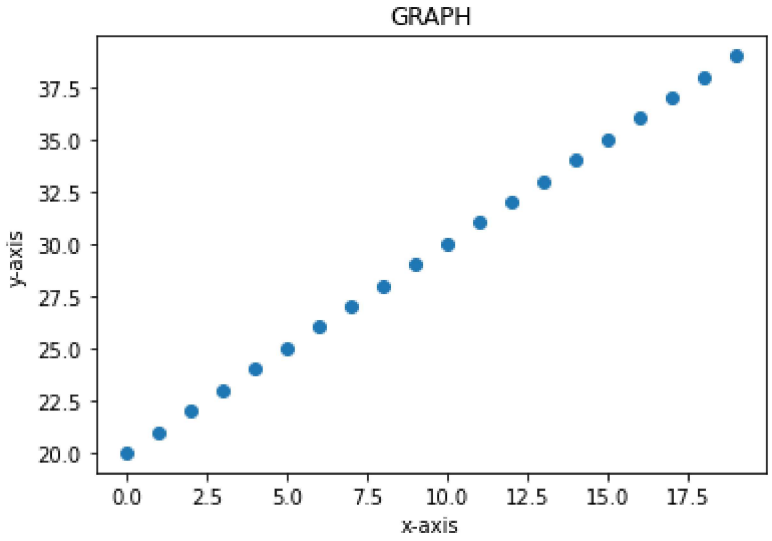
In [146]:
```
plt.scatter(x,y)
```

```
<matplotlib.collections.PathCollection at 0x209845dcda0>
```

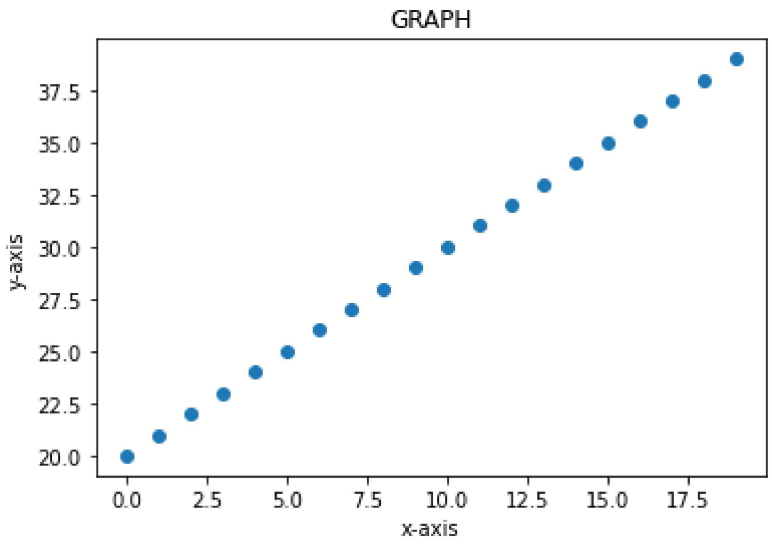


In [147]:
```
plt.scatter(x,y)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('GRAPH')
```

```
Text(0.5, 1.0, 'GRAPH')
```

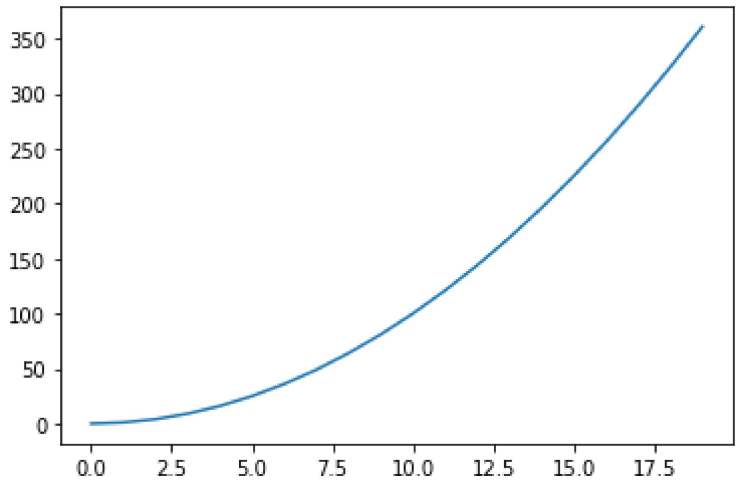


In [148]:
```
plt.scatter(x,y)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('GRAPH')
plt.savefig('Graph.png')
```



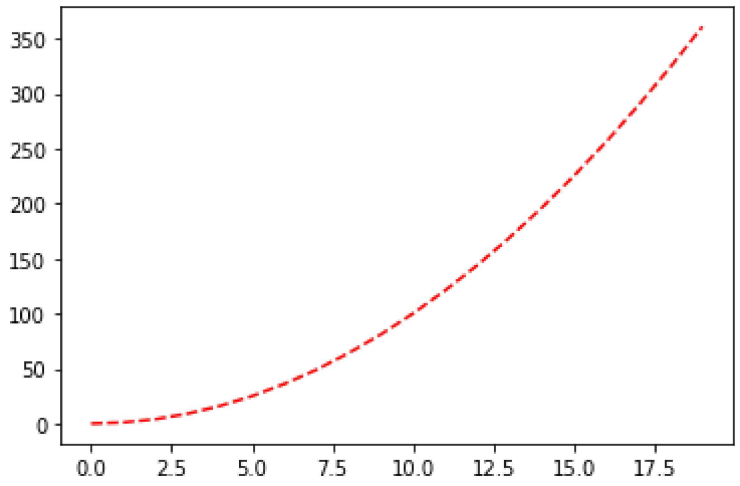
In [149]:
```
y = x**2
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x20984a17a90>]
```



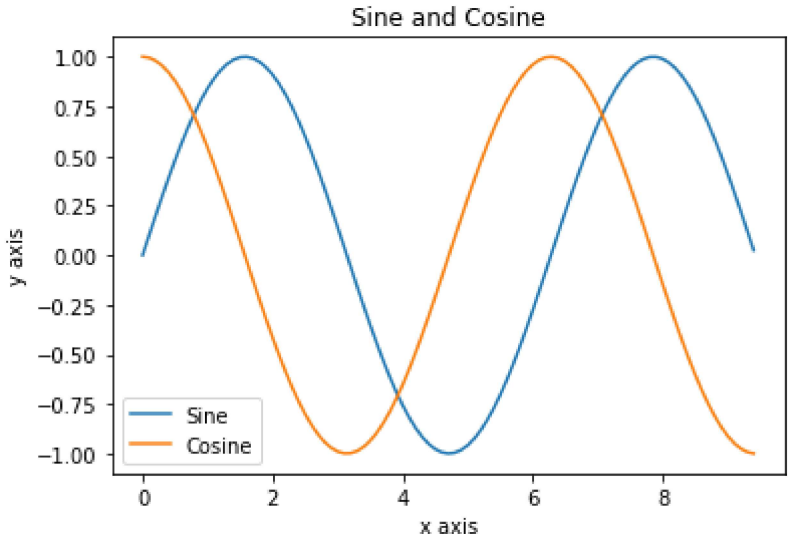
In [150]:
```
plt.plot(x,y,'r--')
```

```
[<matplotlib.lines.Line2D at 0x20984a87be0>]
```



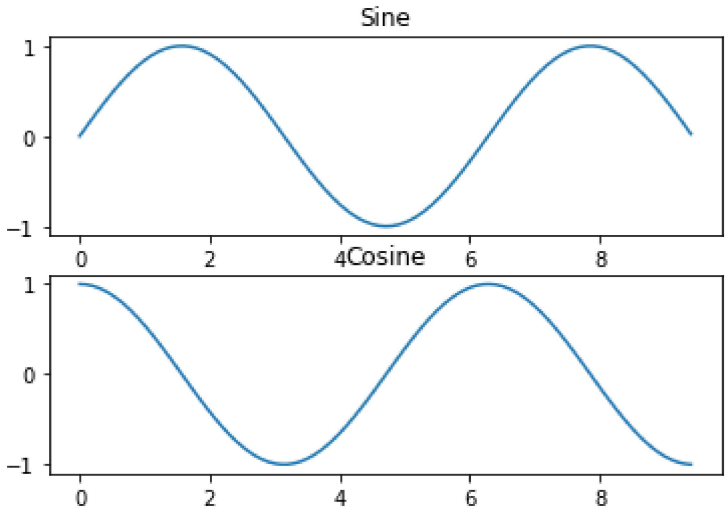
In [151]:
```
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)


plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
```

```
<matplotlib.legend.Legend at 0x20984aeae80>
```



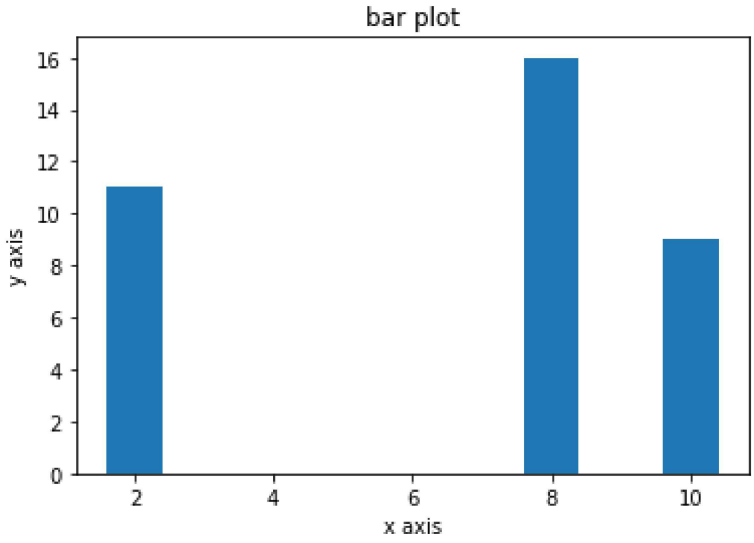
In [152]:
```
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2,1, 1)
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2,1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')
```

```
Text(0.5, 1.0, 'Cosine')
```



In [153]:
```
x=[2,8,10]
y=[11,16,9]
plt.bar(x,y)
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('bar plot')
```

```
Text(0.5, 1.0, 'bar plot')
```



In [ ]:

In [ ]:

In [ ]: