Outline

1. Introduction to Strings
2. Numbers
3. Arithmetic Operations
4. Using Help
5. Booleans
6. Datetime Module
7. if-then-else Statements
8. Functions
9. Sets
10. Lists
11. Dictionaries
12. Tuples
13. Log
14. Recursion, Fibonacci Sequence and Memoization
15. Random Number Module
16. List Comprehension
17. Classes and Objects

In [1]:
```
print("Hello World!")
```

```
Hello World!
```

Introduction to strings

In [2]:
```
message = "Meet me tonight."
print(message)
```

```
Meet me tonight.
```

In [3]:
```
message2 = 'The clock strikes at midnight'
print(message2)
```

```
The clock strikes at midnight
```

In [4]:
```
message3 = 'I\'m looking for someone to share advanture'
print(message3)
```

```
I'm looking for someone to share advanture
```

In [5]:
```
#For more than one line with quotes
movie_quote = """One of my favourite lines from Godfather is:
        "I'm going to make him offer he can't refuse."
        Do you know who said this?"""
print(movie_quote)
```

```
One of my favourite lines from Godfather is:
        "I'm going to make him offer he can't refuse."
        Do you know who said this?
```

Numbers

In [6]:
```
#Types of numbers: int, float, complex
a = 496 #This is a perfext number
print('Type of a number ', type(a))
print('Value of a number ', a)
```

```
Type of a number  <class 'int'>
Value of a number  496
```

In [7]:
```
#floating point
e = 2.718281828
print('Type of a number ', type(e))
print('Value of a number ', e)
```

```
Type of a number  <class 'float'>
Value of a number  2.718281828
```

In [8]:
```
#complex number
z = 2 - 6.1j
print(z)
print("Real part of z is ", z.real)
print("Imaginary part of z is ", z.imag)
```

```
(2-6.1j)
Real part of z is  2.0
Imaginary part of z is  -6.1
```

Arithmetic

- Numbers: int, floats, complex
- Operations: +, -, /, *

In [9]:
```
x = 28 # int
y = 28.0 # float
```

In [10]:
```
# To convert number into complex number
a = 1.732 #float
print(a)
```

```
1.732
```

In [11]:
```
a = 1.732 + 0j
print(a)
```

```
(1.732+0j)
```

In [12]:
```
# Or pass number to complex constructor
print(complex(1.732))
```

```
(1.732+0j)
```

In [13]:
```
# Arithmetic operations
a = 2 # int
b = 6.0 # float
c = 12 + 0j # complex number
```

In [14]:
```
# Rule: widen numbers so they are of same type
# Addition
s = a + b # int + float = float (a is widen to float)
print(s)
```

```
8.0
```

In [15]:
```
# Subtraction
ss = b - a # float - int = float (a is widen to float)
print(ss)
```

```
4.0
```

In [16]:
```
# Multiplication
m = a * 7 # int * int
print(m)
```

```
14
```

In [17]:
```
# Division
dd = c / b # complex / float (b is widen to complex)
print(dd)
```

```
(2+0j)
```

In [18]:
```
rem = 16 % 5 # This gives remainder
print(rem)
```

```
1
```

In [19]:
```
rem = 16 % 5 # This gives remainder
print(rem)
```

```
1
```

Using Help in Python

In [20]:
```
# Accessing help in python
help(pow)
```

```
Help on built-in function pow in module builtins:

pow(x, y, z=None, /)
    Equivalent to x**y (with two arguments) or x**y % z (with three arg
uments)

    Some types, such as ints, are able to use a more efficient algorith
m when
    invoked using the three argument form.
```

Booleans

- In python every string is converted to true and only empty string is converted to false.
- Trivial values are converted to true and non-trivial values are false.

In [21]:
```
a = 3
b = 5
# == for comparison
print(a == b)
```

```
False
```

In [22]:
```
print(a != b)
```

```
True
```

datetime module

In [23]:
```
import datetime

# Default format for date is yyyy-mm-dd
gvr = datetime.date(1956, 1, 31)
print(gvr)
print(gvr.year)
print(gvr.month)
print(gvr.day)
```

```
1956-01-31
1956
1
31
```

In [24]:
```
# To add or subtract number of days from date use timedelta
mill = datetime.date(2000, 1, 1)
# This takes number of days as arg
dt = datetime.timedelta(100)
# Positive number will increase the date and negative number will decrea
se the date
print("mill + dt : ",mill + dt)
```

```
mill + dt :  2000-04-10
```

In [25]:
```
# Reformat the date
# Day-name, Month-name day #, Year
# Two ways to do this
print(gvr.strftime("%A, %B %d, %Y"))
message = "GVR was born on  {:%A, %B %d, %Y}."
print(message.format(gvr))
```

```
Tuesday, January 31, 1956
GVR was born on  Tuesday, January 31, 1956.
```

In [26]:
```
# Format similar to launch date
launch_date = datetime.date(2017, 3, 30)
launch_time = datetime.time(22, 27, 0)
launch_datetime = datetime.datetime(2017, 3, 30, 22, 27, 0)
print(launch_date)
print(launch_time)
print(launch_datetime)
```

```
2017-03-30
22:27:00
2017-03-30 22:27:00
```

In [27]:
```
# Access current date and time
# This can be achieved using method today
now = datetime.datetime.today()
print(now)
```

```
2019-02-07 22:51:56.032523
```

In [28]:
```
# Taking date as a string and converting that string as a date
# This can be done using method String parse Time a.k.a strptime
moon_landing = "7/20/1969"
moon_landing_datetime = datetime.datetime.strptime(moon_landing, "%m/%d
/%Y")
print(moon_landing_datetime)
print(type(moon_landing_datetime))
```

```
1969-07-20 00:00:00
<class 'datetime.datetime'>
```

if-else

In [29]:
```
#1
# Collect String / Test Length

password = input("Please Enter a String : ")

if len(password) < 6:
        print("Your String is too short.")
        print("Enter a String with at least 6 characters.")
```

```
Please Enter a String : asdf
Your String is too short.
Enter a String with at least 6 characters.
```

In [30]:
```
#2
# Prompt the user to enter number / test if ecen or odd
num = int(input("Enter a number : "))

if num % 2 == 0:
        print("Number is even.")
else:
        print("Number is odd.")
```

```
Enter a number : 4
Number is even.
```

In [31]:
```
#3
# Scalane Triangle : All sides have different sides
# Isosceles Triangle : Two sides have same length
# Equilateral Triangle : All sides are equal.

a = int(input("Enter length of side a : "))
b = int(input("Enter length of side b : "))
c = int(input("Enter length of side c : "))

if a!=b and b!=c and a!=c:
        print("This is a scalane triangle.")
elif a == b and b == c:
        print("This is equilateral triangle.")
else:
        print("It is Isosceles triangle.")
```

```
Enter length of side a : 4
Enter length of side b : 4
Enter length of side c : 4
This is equilateral triangle.
```

Functions

Types of Arguments

1. Keyword arguments which are **with** some initial value
2. Required arguments which are **without** any initial values

In [32]:
```
import math

def ping():
    return "Ping!"

def volume(r):
    """Returns the volume of sphere with radius r."""
    v = (4.0/3.0) * math.pi * r**3
    return v

def traingle_area(b, h):
    """This function returns the area of the traingle"""
    return 0.5 * b * h
```

In [33]:
```
x = ping()
print(x)

vol = volume(3)
print(vol)

area_tr = traingle_area(2, 4)
print(area_tr)
```

```
Ping!
113.09733552923254
4.0
```

In [34]:
```
# function with keyword arguments
def cm(feet = 0, inches = 0):
        """converts a length from feet and inches to centimeters"""
        inches_to_cm = inches * 2.54
        feet_to_cm = feet * 12 * 2.54
        return inches_to_cm + feet_to_cm

print(cm(feet = 5))
print(cm(inches = 70))
print(cm(feet = 5, inches = 8))
print(cm(inches = 8, feet = 5))
```

```
152.4
177.8
172.72
172.72
```

While writing a function with combination of required and keyword arguments, **Keyword arguments are last** and **Keyword arguments are passed by names**

In [35]:
```
def g(y, x = 0):
        return x + y

print(g(5))
print(g(5, x = 4))
```

```
5
9
```

Sets

- Sets do not contain duplicate elements

Adding element to the set

In [36]:
```
example = set()
example.add(42)
example.add(False)
example.add(3.14159)
print(example)
print("Length of set : ",len(example))
```

```
{False, 42, 3.14159}
Length of set :  3
```

Removing element from the set

In [37]:
```
print("Removing element 42 from the set")
example.remove(42)
print(example)
```

```
Removing element 42 from the set
{False, 3.14159}
```

In [38]:
```
# element can also be removed using discard method
print("Adding element 50 to the set")
example.add(50)
print(example)

print("After discarding the element 50 from the set")
example.discard(50)
print(example)
```

```
Adding element 50 to the set
{False, 50, 3.14159}
After discarding the element 50 from the set
{False, 3.14159}
```

Prepoulate set

In [39]:
```
example2 = set([28, True, 2.71828, "Helium"])
print(len(example2))
print(example2)
```

```
4
{'Helium', True, 2.71828, 28}
```

Remove all the elements from the set

In [40]:
```
# To remove all the elements from the set use clear method
example2.clear()
print(len(example2))
```

```
0
```

Set Operations

In [41]:
```
# Union and intersection of set
# Integers 1 - 10

odds = set([1, 3, 5, 7, 9])
evens = set([2, 4, 6, 8, 10])
primes = set([2, 3, 5, 7])
composites = set([4, 6, 8, 9, 10])
```

In [42]:
```
print("union of odds and evens")
print(odds.union(evens))
print("Intersection of odds and primes")
print(odds.intersection(primes))
print("Intersection of evens and primes")
print(primes.intersection(evens))
print("Union of primes and composite")
print(primes.union(composites))
print(2 in primes)
print(9 in evens)
```

```
union of odds and evens
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Intersection of odds and primes
{3, 5, 7}
Intersection of evens and primes
{2}
Union of primes and composite
{2, 3, 4, 5, 6, 7, 8, 9, 10}
True
False
```

Lists

In [43]:
```
# Two ways to create lists in python
example = list()
example2 = []
```

In [44]:
```
primes = [2, 3, 5, 7, 11, 13]
primes.append(17)
primes.append(19)
print(primes)
```

```
[2, 3, 5, 7, 11, 13, 17, 19]
```

In [45]:
```
print(primes[0])
print("Looking at the last item")
print(primes[-1])
```

```
2
Looking at the last item
19
```

In [46]:
```
# Slicing
s1 = primes[2:5]
print(s1)
```

```
[5, 7, 11]
```

In [47]:
```
s2 = primes[0:6]
print(s2)
```

```
[2, 3, 5, 7, 11, 13]
```

In [48]:
```
example = [128, True, "Alpha", 1.732, [64, False]]
print(example)
```

```
[128, True, 'Alpha', 1.732, [64, False]]
```

In [49]:
```
# Combining lists
numbers = [1, 2, 3]
letters = ['a', 'b', 'c']
print("After Combining")
print(numbers + letters)
print(letters + numbers)
```

```
After Combining
[1, 2, 3, 'a', 'b', 'c']
['a', 'b', 'c', 1, 2, 3]
```

In [50]:
```
print("To reverse the list")
numbers.reverse()
print(numbers)
```

```
To reverse the list
[3, 2, 1]
```

Dictionaries

In [51]:
```
# FriendFace Post
# user_id = 209
# message = "I love my family."
# language = "English"
# datetime = "20230215T124231Z"
# location = (44.590533, -104.715556)

post = {"user_id": 209, "message": "I love my family.", "language":"Engl
ish",
                "datetime":"20230215T124231Z", "location": (44.590533, -
104.715556)}
```

In [52]:
```
print(type(post))
print(post)
```

```
<class 'dict'>
{'user_id': 209, 'message': 'I love my family.', 'language': 'English',
'datetime': '20230215T124231Z', 'location': (44.590533, -104.715556)}
```

In [53]:
```
post2 = dict(message = "I love you too.", language = "English")
post2["user_id"] = 209
post2["datetime"] = "19771116T093001Z"
print(post2)
```

```
{'message': 'I love you too.', 'language': 'English', 'user_id': 209,
'datetime': '19771116T093001Z'}
```

In [54]:
```
# To access specific key with value
print(post['message'])
```

```
I love my family.
```

In [55]:
```
# To avoid error that key is not in the dictionary
if 'location' in post2:
        print(post2['location'])
else:
        print("The post does not contain location value.")
```

```
The post does not contain location value.
```

In [56]:
```
# Another way to do this is using try and except

try:
        print(post2['location'])
except:
        print("The post does not have location value.")
```

```
The post does not have location value.
```

In [57]:
```
# Use get() method to get the key value
loc = post2.get('location', None)
print(loc)
```

```
None
```

In [58]:
```
# To iterate over all the keys
for keys in post.keys():
        value = post[keys]
        print(keys, "=", value)
```

```
user_id = 209
message = I love my family.
language = English
datetime = 20230215T124231Z
location = (44.590533, -104.715556)
```

In [59]:
```
print("Another Method")
# Another way to iterate over all the key value pairs
for key, values in post.items():
        print(key, "=", values)
```

```
Another Method
user_id = 209
message = I love my family.
language = English
datetime = 20230215T124231Z
location = (44.590533, -104.715556)
```

Tuples

In [60]:
```
import sys
import timeit

# Difference between list and tuples
# List Example
prime_numbers = [2, 3, 5, 7, 11, 13, 17]

# Tuple Example
perfect_squares = (1, 4, 9, 16, 25, 36)
```

In [61]:
```
# Display Length
print("# Primes = ", len(prime_numbers))
print("# Squares = ", len(perfect_squares))
```

```
# Primes =  7
# Squares =  6
```

In [62]:
```
# Iterate over both sequences
for p in prime_numbers:
        print("Prime : ", p)

for n in perfect_squares:
        print("Square : ", n)
```

```
Prime :  2
Prime :  3
Prime :  5
Prime :  7
Prime :  11
Prime :  13
Prime :  17
Square :  1
Square :  4
Square :  9
Square :  16
Square :  25
Square :  36
```

In [63]:
```
# Methods for list
print("List methods")
print(dir(prime_numbers))
print(80*"-")
print("Tuples methods")
print(dir(perfect_squares))
```

```
List methods
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute
__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__in
it__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr_
_', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof
__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count',
'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
----------------------------------------------------------------------
---------
Tuples methods
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__d
oc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem
__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subcla
ss__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmul__', '__se
tattr__', '__sizeof__', '__str__', '__subclasshook__', 'count', 'inde
x']
```

In [64]:
```
# Creating list and tuple of exactly same elemets
list_eg = [1, 2, 3, "a", "b", "c", True, 3.14159]
tuple_eg = (1, 2, 3, "a", "b", "c", True, 3.14159)
print("Getting size of list and tuple")
print("List size = ", sys.getsizeof(list_eg))
print("Tuple size = ", sys.getsizeof(tuple_eg))
```

```
Getting size of list and tuple
List size =  128
Tuple size =  112
```

In [65]:
```
# To see which takes more time list or tuples
list_test = timeit.timeit(stmt = "[1, 2, 3, 4, 5]", number = 10000000)
tuple_test = timeit.timeit(stmt = "(1, 2, 3, 4, 5)", number = 10000000)
print("List time : ", list_test)
print("Tuple time : ", tuple_test)
```

```
List time :  0.5954892069967173
Tuple time :  0.1480178630008595
```

Differences between lists and tuples:

1. Lists takes more momory than tuples
2. You can add, remove or change data in lists, while tuples cannot be changed
3. Tuples can be made more quickly then lists
4. Tuples are immutable (unable to change)

In [66]:
```
# Empty tuple
empty_tuple = ()
# Tuple containing only one element is considered string
test1 = ("a")
# To make a tuple with one element
test11 = ("a",)
test2 = ("a", "b")
test3 = ("a", "b", "c")
print(empty_tuple)
print(test1)
print(test11)
print(test2)
print(test3)
```

```
()
a
('a',)
('a', 'b')
('a', 'b', 'c')
```

In [67]:
```
# Alternative for construction of tuples
tt1 = 1,
tt2 = 1, 2
tt3 = 1, 2, 3
print(tt1)
print(tt2)
print(tt3)
print(type(tt1))
print(type(tt2))
print(type(tt3))
```

```
(1,)
(1, 2)
(1, 2, 3)
<class 'tuple'>
<class 'tuple'>
<class 'tuple'>
```

In [68]:
```
# Tuple with one element
# (age, country, knows_python)
survey = (27, "Vietnam", True)
age = survey[0]
country = survey[1]
knows_python = survey[2]

print("Age = ", age)
print("Country = ", country)
print("Knows Python?", knows_python)
```

```
Age =  27
Country =  Vietnam
Knows Python? True
```

In [69]:
```
survey2 = (21, "Switzerland", False)
age, country, knows_python = survey2
print("Age = ", age)
print("Country = ", country)
print("Knows_Python?", knows_python)
```

```
Age =  21
Country =  Switzerland
Knows_Python? False
```

In [70]:
```
country = ("Australia",)
print(country)
```

```
('Australia',)
```

- Number of variable should be equal to number of values

Logging

- logging module in python gives Progress Reports of the code
- Purpose: Record Progress and problems ...
- Levels : Debug, Info, Warning, Error and Critical

In [71]:
```
import logging

print(dir(logging))
```

```
['BASIC_FORMAT', 'BufferingFormatter', 'CRITICAL', 'DEBUG', 'ERROR', 'F
ATAL', 'FileHandler', 'Filter', 'Filterer', 'Formatter', 'Handler', 'IN
FO', 'LogRecord', 'Logger', 'LoggerAdapter', 'Manager', 'NOTSET', 'Null
Handler', 'PercentStyle', 'PlaceHolder', 'RootLogger', 'StrFormatStyl
e', 'StreamHandler', 'StringTemplateStyle', 'Template', 'WARN', 'WARNIN
G', '_STYLES', '_StderrHandler', '__all__', '__author__', '__builtins_
_', '__cached__', '__date__', '__doc__', '__file__', '__loader__', '__n
ame__', '__package__', '__path__', '__spec__', '__status__', '__version
__', '_acquireLock', '_addHandlerRef', '_checkLevel', '_defaultFormatte
r', '_defaultLastResort', '_handlerList', '_handlers', '_levelToName',
'_lock', '_logRecordFactory', '_loggerClass', '_nameToLevel', '_release
Lock', '_removeHandlerRef', '_showwarning', '_srcfile', '_startTime',
'_warnings_showwarning', 'addLevelName', 'atexit', 'basicConfig', 'capt
ureWarnings', 'collections', 'critical', 'currentframe', 'debug', 'disa
ble', 'error', 'exception', 'fatal', 'getLevelName', 'getLogRecordFacto
ry', 'getLogger', 'getLoggerClass', 'handlers', 'info', 'io', 'lastReso
rt', 'log', 'logMultiprocessing', 'logProcesses', 'logThreads', 'makeLo
gRecord', 'os', 'raiseExceptions', 'root', 'setLogRecordFactory', 'setL
oggerClass', 'shutdown', 'sys', 'threading', 'time', 'traceback', 'war
n', 'warning', 'warnings', 'weakref']
```

In [72]:
```
# Create and Configure logger
# To fix this we need to uypdate basic config call and set level to DEBU
G in line 10.

# Changing the format of the log
LOG_FORMAT = "%(levelname)s %(asctime)s - %(message)s"
# To override the file every time change the file mode to 'w'.
logging.basicConfig(filename = "./Log_example.log",
        level = logging.DEBUG,
        format = LOG_FORMAT,
        filemode = 'w')
# Create logger object and call getLogger()
# Logger without the name is called root logger
logger = logging.getLogger()
```

In [73]:
```
# Test the logger
logger.info("Our Second Message")
```

In [81]:
```
"""
What does this logger level means:

Level                    Numeric Value
------------------------------
NOTSET                   0
DEBUG                    10
INFO                     20
WARNING                  30
ERROR                    40
CRITICAL                 50
------------------------------

Loggers only write the value greater than or equal to set level.
No value is written in the log file as the default value of root logger
 is 30.

"""

# Get the level of the logger
print(logger.level)
```

```
10
```

In [82]:
```
# Creating logger message with all five levels
logger.debug("This is a harmless debug message.")
logger.info("Just some useful info.")
logger.warning("I'm sorry, but I can't do that, Jimmy.")
logger.error("Did you just try to divide by zero?")
logger.critical("The entire internet is down!!")
```

Example for Logging

In [84]:
```
import math

def quadratic_formula(a, b, c):
        """Returns the solution to the quadratic equations ax^2 + bx + c
 = 0."""
        logger.info("quadratic_formula({0}, {1}, {2})".format(a, b, c))

        # Compute the discriminant
        logger.debug("# Compute the discriminant")
        disc = b**2 - 4*a*c

        # Compute the two roots
        logger.debug("# Compute the two roots")
        root1 = (-b + math.sqrt(disc)) / (2*a)
        root2 = (-b - math.sqrt(disc)) / (2*a)

        # Return the roots
        logger.debug("# Return the roots")
        return (root1, root2)
```

In [85]:
```
roots = quadratic_formula(1, 0, -4)
print(roots)
```

```
(2.0, -2.0)
```

In [88]:
```
import os
location = './'
for filename in os.listdir(location):
    if filename == 'Log_example.log':
       f  = open('Log_example.log', "r")
       print (f.read())

# The number after comma after time is millisecond portion of the time
```

```
INFO 2019-02-07 22:50:47,222 - Our Second Message
DEBUG 2019-02-07 22:50:47,231 - This is a harmless debug message.
INFO 2019-02-07 22:50:47,231 - Just some useful info.
WARNING 2019-02-07 22:50:47,231 - I'm sorry, but I can't do that, Jimm
y.
ERROR 2019-02-07 22:50:47,231 - Did you just try to divide by zero?
CRITICAL 2019-02-07 22:50:47,231 - The entire internet is down!!
INFO 2019-02-07 22:50:47,252 - quadratic_formula(1, 0, -4)
DEBUG 2019-02-07 22:50:47,253 - # Compute the discriminant
DEBUG 2019-02-07 22:50:47,253 - # Compute the two roots
DEBUG 2019-02-07 22:50:47,253 - # Return the roots
```

Recursion, Fibinacci Sequence and Memoization

In [89]:
```
def fibonacci(n):
        # Check if the input is an integer
        if type(n) != int:
                raise TypeError("n must be a positive integer")
        if n < 1:
                raise ValueError("n must be a positive integer")


        if n == 1:
                return 1
        elif n == 2:
                return 1
        elif n > 2:
                return fibonacci(n-1) + fibonacci(n-2)
```

In [101]:
```
# This slow

for i in range(1, 31):
        print(i, " : ", fibonacci(i))
```

```
1  :  1
2  :  1
3  :  2
4  :  3
5  :  5
6  :  8
7  :  13
8  :  21
9  :  34
10  :  55
11  :  89
12  :  144
13  :  233
14  :  377
15  :  610
16  :  987
17  :  1597
18  :  2584
19  :  4181
20  :  6765
21  :  10946
22  :  17711
23  :  28657
24  :  46368
25  :  75025
26  :  121393
27  :  196418
28  :  317811
29  :  514229
30  :  832040
```

To make this faster use Memoization

Idea: Cache values ---> Store the values for the recent function call so future calls do not have to repeat the work

Implementation of Memoization can be done in several ways

1. Implement Explicitly

In [102]:
```
# Create a dictionary called fibonacci cache
fibonacci_cache = {}
# Rewriting the function to make use of cache values
def fibonacci2(n):
        # If we have cached the value, then return it
        if n in fibonacci_cache:
                return fibonacci_cache[n]

        # Compute the Nth term
        if n == 1:
                value = 1
        elif n == 2:
                value = 2
        elif n > 2:
                value = fibonacci2(n - 1) + fibonacci2(n - 2)

        # Canhe the value and return it
        fibonacci_cache[n] = value
        return value
```

In [103]:
```
for n in range(1, 31):
        print(n , " : ", fibonacci2(n))
```

```
1  :  1
2  :  2
3  :  3
4  :  5
5  :  8
6  :  13
7  :  21
8  :  34
9  :  55
10  :  89
11  :  144
12  :  233
13  :  377
14  :  610
15  :  987
16  :  1597
17  :  2584
18  :  4181
19  :  6765
20  :  10946
21  :  17711
22  :  28657
23  :  46368
24  :  75025
25  :  121393
26  :  196418
27  :  317811
28  :  514229
29  :  832040
30  :  1346269
```

2. Using built-in tools

In [104]:
```
# import Least Recently used cache
from functools import lru_cache

@lru_cache(maxsize = 1000)
def fibonacci3(n):
        # Check if the input is an integer
        if type(n) != int:
                raise TypeError("n must be a positive integer")
        if n < 1:
                raise ValueError("n must be a positive integer")


        if n == 1:
                return 1
        elif n == 2:
                return 1
        elif n > 2:
                return fibonacci(n-1) + fibonacci(n-2)
```

In [105]:
```
for i in range(1, 31):
        print(i, " : ", fibonacci3(i))
```

```
1  :  1
2  :  1
3  :  2
4  :  3
5  :  5
6  :  8
7  :  13
8  :  21
9  :  34
10  :  55
11  :  89
12  :  144
13  :  233
14  :  377
15  :  610
16  :  987
17  :  1597
18  :  2584
19  :  4181
20  :  6765
21  :  10946
22  :  17711
23  :  28657
24  :  46368
25  :  75025
26  :  121393
27  :  196418
28  :  317811
29  :  514229
30  :  832040
```

Random Number Generator

WARNING: The pseudo-random generators of this module should not be used for security purposes. Use os.urandom() or SystemRandom if you require a cryptographically secure pseudo-random generator.

In [107]:
```
import random

# Display 10 random numbers from interval [0, 1)
for i in range(10):
        print(random.random())
```

```
0.03105817649777798
0.5222125120387405
0.6630708112162361
0.7039406552917004
0.7373647312464238
0.9287269012591494
0.6453285514089452
0.366462196153734
0.1744825024676444
0.05508612884773967
```

In [108]:
```
# Generate random numbers from interval [3, 7)
"""
Several ways of doing this
call random():                                  # in [0,1)
scale number (multiply by 4):   # in [0,4) (as difference between 7 and
 3 is 4)
Shift number (add 3):                   # in [3,7)

Reason for doing this to show random() function can be used to build
customized random number generator.
"""
def my_random():
        # Random, scale, shift, return ...
        return 4*random.random() + 3

for i in range(10):
        print(my_random())
```

```
3.220624131039197
6.4109627128681215
3.470562406721209
4.454433324097832
6.6901054928969685
4.602889744528509
3.093318843338757
4.507150584240801
5.417477205804376
3.894656007384867
```

In [109]:
```
"""
Easier way to generate random number within specific range is
using uniform() available in random module
"""
for i in range(10):
        print(random.uniform(3,7))


# random() and uniform() are both uniform distributions.
```

```
5.84408285714529
6.469141831457926
5.621257281866029
3.876356217730541
5.0743434152538605
4.373127760905087
4.611349215692316
5.828731821634363
5.913455587938984
4.220903104151846
```

In [110]:
```
# To generate number from bell curve along some mean and standard deviat
ion
# use normalvariate()
for i in range(20):
        print(random.normalvariate(0, 1))
```

```
0.1544326762245482
-0.15363169094948506
-1.0288554259620473
-0.8945491445032854
-0.19419257603681375
0.4178596392355029
-1.0908238303647024
-1.8513427944906267
1.0789463424797916
-2.298558156394942
-0.1131931197304963
-0.3825066027610636
-0.5982720403331815
-1.6143412293910089
0.1821612048362713
0.6147274203119967
0.43748985767244025
0.6055834946642484
-1.0670582073244104
0.8550174292057574
```

In [111]:
```
# Generate numbers within specific range with discrete probability distr
ibutions
# This can be achieved using randint()
for i in range(20):
        print(random.randint(1, 6))
```

```
6
4
4
4
6
3
3
1
3
3
5
4
2
4
1
3
6
5
1
2
```

In [113]:
```
"""
Random selection from list of values. playing rock, paper, scissors.
"""
outcomes = ['rock', 'paper', 'scissors']
# To pick a random values from this list use a choice funtion
for i in range(10):
        print(random.choice(outcomes))
```

```
paper
paper
rock
scissors
scissors
rock
rock
scissors
paper
paper
```

List Comprehension

In [114]:
```
squares = []
for i in range(1, 101):
    squares.append(i**2)

print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 28
9, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 96
1, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 18
49, 1936, 2025, 2116, 2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916, 3
025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356,
4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 5776, 5929, 6084,
6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100,
8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 9801, 10000]
```

In [115]:
```
# List Comprehension
squares2 = [i**2 for i in range(1, 101)]
print(squares2)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 28
9, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 96
1, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 18
49, 1936, 2025, 2116, 2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916, 3
025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356,
4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 5776, 5929, 6084,
6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100,
8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 9801, 10000]
```

In [116]:
```
remainders = [x**2 % 5 for x in range(1, 101)]
print(remainders)
```

```
[1, 4, 4, 1, 0, 1, 4, 4, 1, 0, 1, 4, 4, 1, 0, 1, 4, 4, 1, 0, 1, 4, 4,
1, 0, 1, 4, 4, 1, 0, 1, 4, 4, 1, 0, 1, 4, 4, 1, 0, 1, 4, 4, 1, 0, 1, 4,
4, 1, 0, 1, 4, 4, 1, 0, 1, 4, 4, 1, 0, 1, 4, 4, 1, 0, 1, 4, 4, 1, 0, 1,
4, 4, 1, 0, 1, 4, 4, 1, 0, 1, 4, 4, 1, 0, 1, 4, 4, 1, 0, 1, 4, 4, 1, 0,
1, 4, 4, 1, 0]
```

In [118]:
```
movies = ["Star Wars", "Gandhi", "Casablancs", "Shawshank", "Good Will H
unting", "Raiders of the Lost Ark", "Groundhog Day"]
movies
```

Out[118]:
```
['Star Wars',
 'Gandhi',
 'Casablancs',
 'Shawshank',
 'Good Will Hunting',
 'Raiders of the Lost Ark',
 'Groundhog Day']
```

In [121]:
```
# Getting all the movies that starts from 'G'
print("Without List Comprehension")
gmovies = []
for title in movies:
    if title.startswith("G"):
        gmovies.append(title)

print(gmovies)
```

```
Without List Comprehension
['Gandhi', 'Good Will Hunting', 'Groundhog Day']
```

In [122]:
```
print("With list compression")
gmovies2 = [title for title in movies if title.startswith("G")]
print(gmovies2)
```

```
With list compression
['Gandhi', 'Good Will Hunting', 'Groundhog Day']
```

In [124]:
```
# List containing tuples
movies3 = [("Citizen Kane", 1941), ("Spirited Away", 2001), ("Gattaca",
1997), ("No Country for Old Men", 2007)]
movies3
```

Out[124]:
```
[('Citizen Kane', 1941),
 ('Spirited Away', 2001),
 ('Gattaca', 1997),
 ('No Country for Old Men', 2007)]
```

In [125]:
```
# Find the movies released befre 2000
pre2k = [title for (title, year) in movies3 if year < 2000]
print(pre2k)
```

```
['Citizen Kane', 'Gattaca']
```

In [126]:
```
# Scalar multiplication using list comprehension
v = [2, -3, 1]
w = [4*x for x in v]
print(w)
```

```
[8, -12, 4]
```

In [127]:
```
# Computing Cartesian Coordinate
A = [1, 3, 5, 7]
B = [2, 4, 6, 8]
cartesian_product = [(a,b) for a in A for b in B]
print(cartesian_product)
```

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

Classes and Objects

In [129]:
```
class User:
    pass

# Creating object of a class
user1 = User()
user1.first_name = "John"
user1.last_name = "Dingell"
# Here first name and last name are called fields of the class User()
print(user1.first_name)
print(user1.last_name)
```

```
John
Dingell
```

In [130]:
```
# Creating second user
user2 = User()
user2.first_name = "Frank"
user2.last_name = "poole"
print(user2.first_name, user2.last_name)
```

```
Frank poole
```

In [131]:
```
# Attaching different fields to different users
user1.age = 37
user2.favourite_book = "2001 : A Space Odyssey"
print(user1.age)
print(user2.favourite_book)
```

```
37
2001 : A Space Odyssey
```