RV College of Engineering®

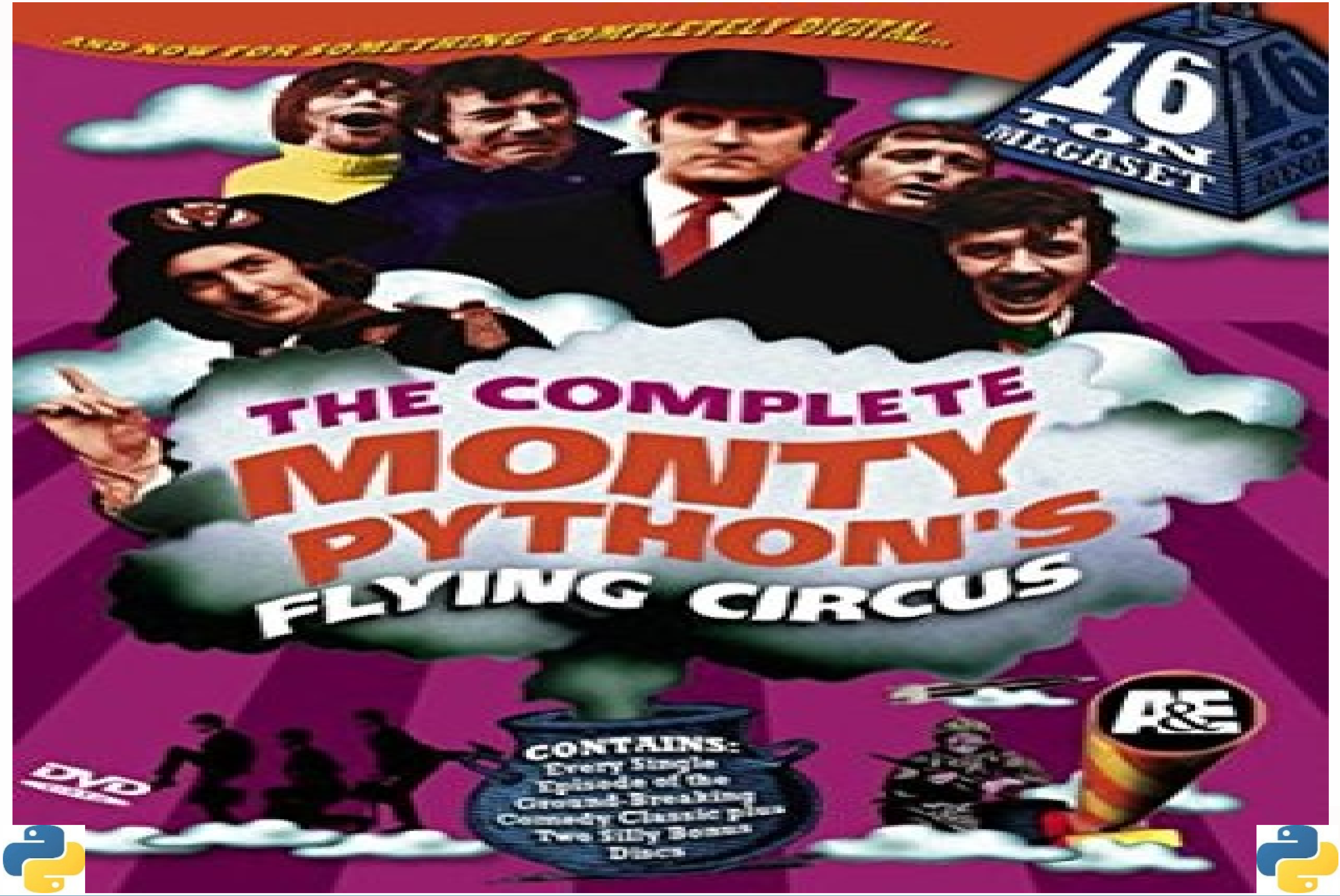
Go, change the world

Object Oriented Programming

Dr. Manjunath M
Assistant Professor,
Dept. of Computer Application
R.V College of Engneering

Introduction to Python

Guido Van Rossum


AND NOW FOR SOMETHING COMPLETELY DIGITAL...
16 TON MEGASET
THE COMPLETE MONTY PYTHON'S FLYING CIRCUS
CONTAINS: Every Single Episode of the Ground-Breaking Comedy Classic Plus Two Silly Bonus Discs
A&E
DVD

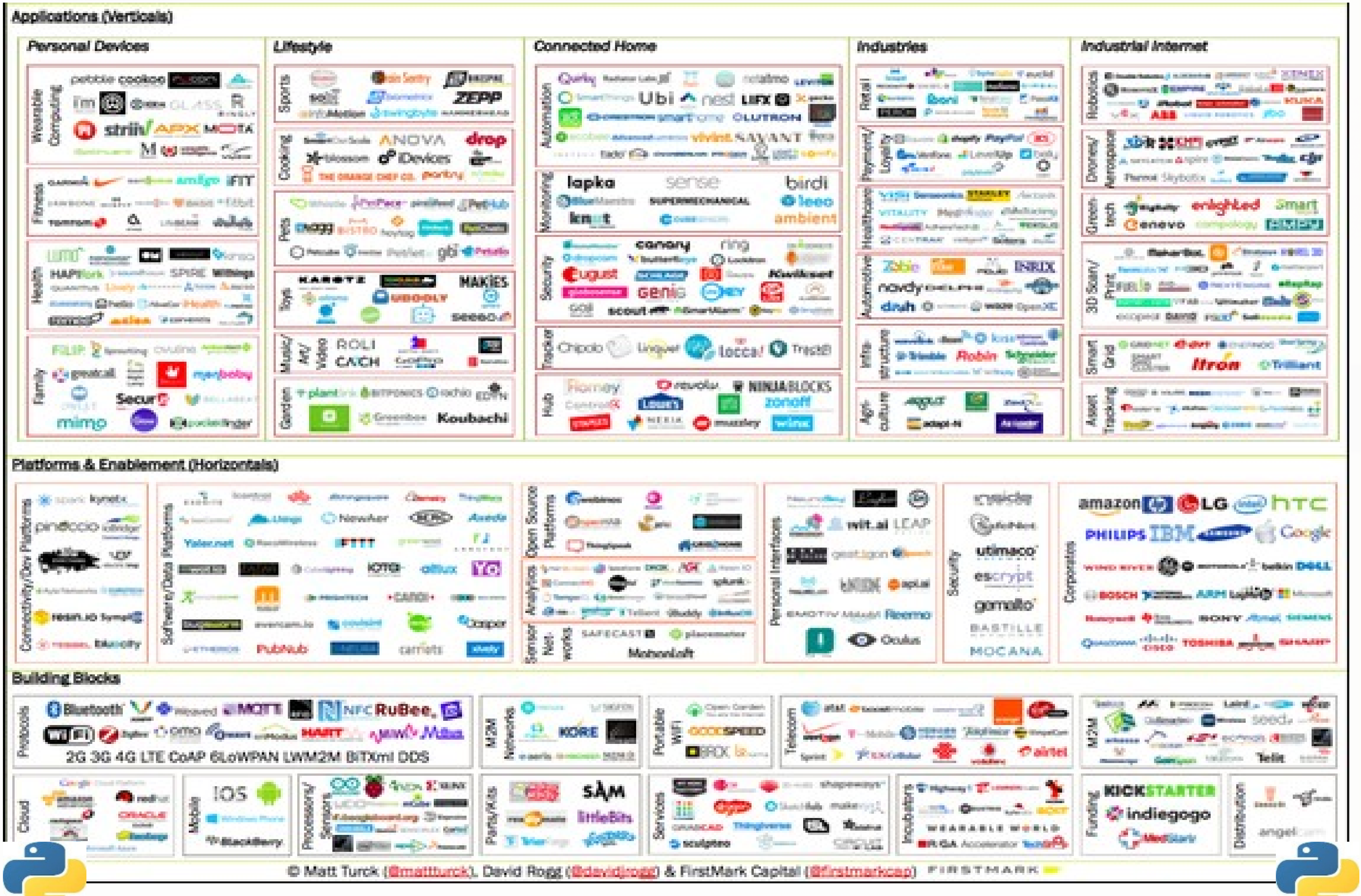
Python Features

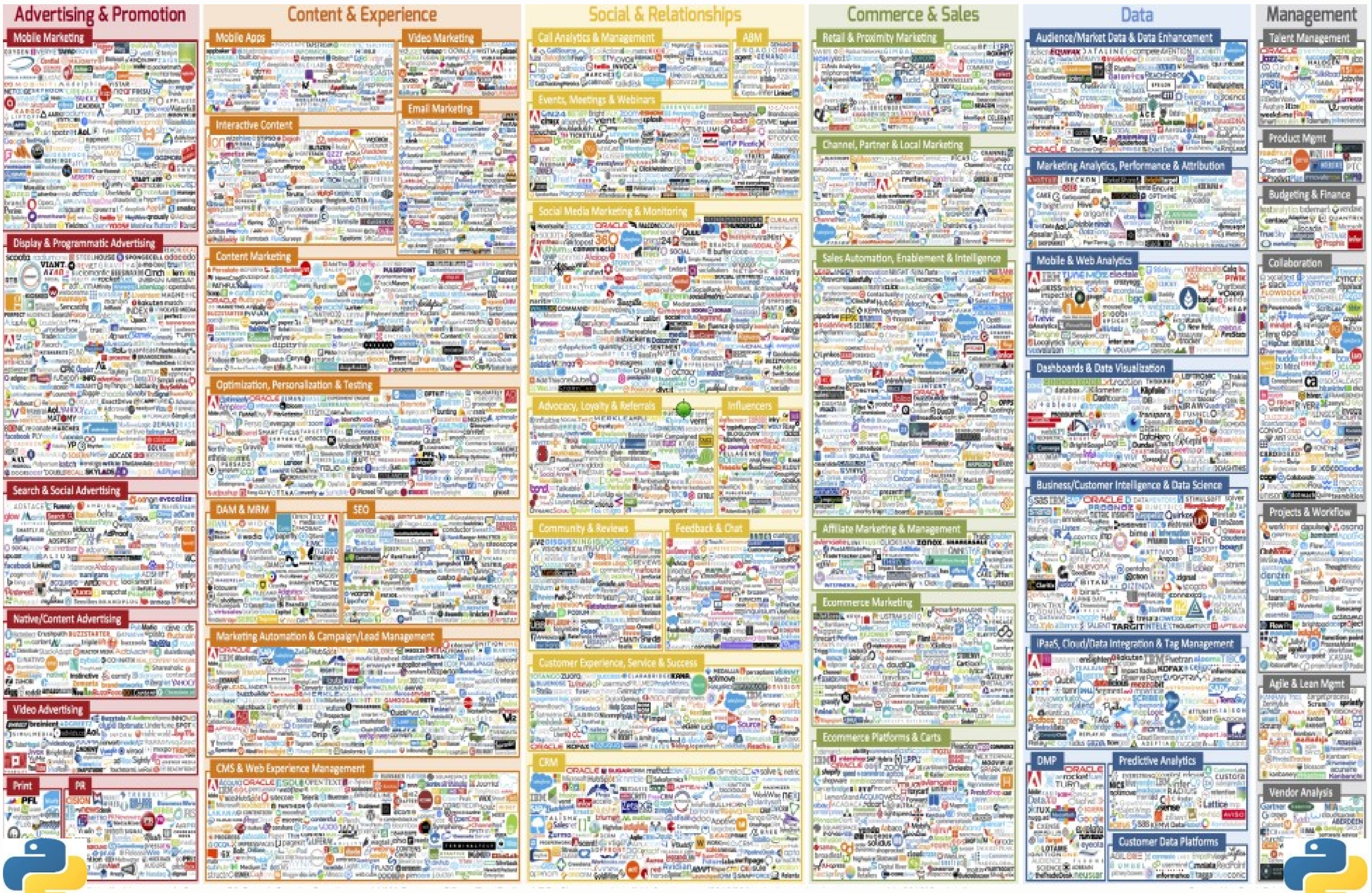
- Scripting Language
- It's free (open source)
- Complete Object-Oriented
- Portable
- Powerful
- A broad standard library
- Mixes good features from Java, Perl and scheme
- Easy to use and Easy to Learn
- Databases

Major Applications of Python

- System Utilities
- Web Applications (Tkinter, gtk,Qt, Windows)
- Internet scripting
- Embedded Scripting
- Artificial Intelligence
- Image Processing
- Networking
- IoT etc….

Why Python?


Applications (Verticals)
Personal Devices
Wearable Computing
Fitness
Health
Family
Lifestyle
Sports
Cooking
Pets
Toys
Music/Art/Video
Garden
Connected Home
Automation
Monitoring
Security
Tracker
Hub
Industries
Retail
Payment/Loyalty
Healthcare
Automotive
Infrastructure
Agriculture
Industrial Internet
Robotics
Drones/Aerospace
Green tech
3D Scan/Print
Smart Grid
Asset Tracking
Platforms & Enablement (Horizontals)
Connectivity/Dev Platforms
Software/Data Platforms
Open Source Platforms
Analytics
Sensor Networks
Personal Interfaces
Security
Corporates
Building Blocks
Protocols
2G 3G 4G LTE CoAP 6LoWPAN LWM2M BITXml DDS
M2M Networks
Portable WiFi
Telecom
M2M
Cloud
Mobile
Processors/Sensors
Parts/Kits
Services
Incubators
Funding
Distribution
© Matt Turck (@mattturck), David Rogg (@davidjrogg) & FirstMark Capital (@firstmarkcap)
FIRSTMARK

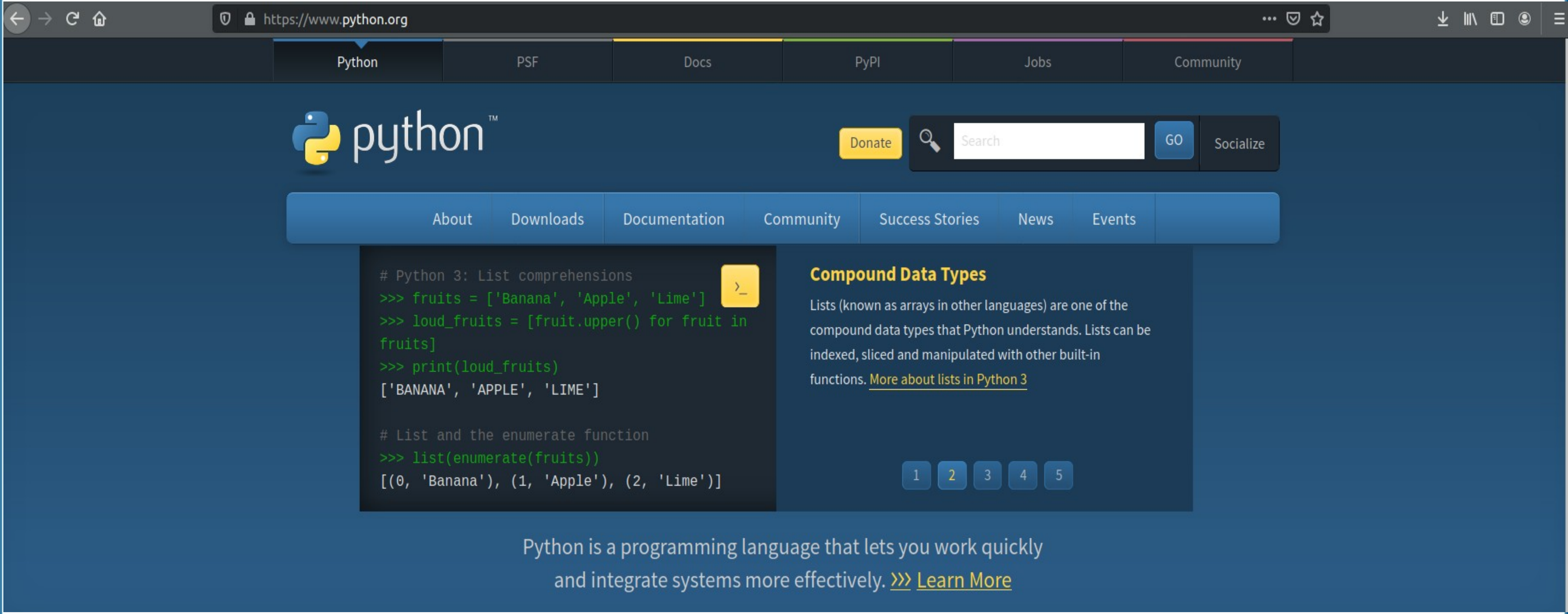

Advertising & Promotion
Mobile Marketing
Display & Programmatic Advertising
Search & Social Advertising
Native/Content Advertising
Video Advertising
Print
PR
Content & Experience
Mobile Apps
Video Marketing
Interactive Content
Email Marketing
Content Marketing
Optimization, Personalization & Testing
DAM & MRM
SEO
Marketing Automation & Campaign/Lead Management
CMS & Web Experience Management
Social & Relationships
Call Analytics & Management
ABM
Events, Meetings & Webinars
Social Media Marketing & Monitoring
Advocacy, Loyalty & Referrals
Influencers
Community & Reviews
Feedback & Chat
Customer Experience, Service & Success
CRM
Commerce & Sales
Retail & Proximity Marketing
Channel, Partner & Local Marketing
Sales Automation, Enablement & Intelligence
Affiliate Marketing & Management
Ecommerce Marketing
Ecommerce Platforms & Carts
Data
Audience/Market Data & Data Enhancement
Marketing Analytics, Performance & Attribution
Mobile & Web Analytics
Dashboards & Data Visualization
Business/Customer Intelligence & Data Science
iPaaS, Cloud/Data Integration & Tag Management
DMP
Predictive Analytics
Customer Data Platforms
Management
Talent Management
Product Mgmt
Budgeting & Finance
Collaboration
Projects & Workflow
Agile & Lean Mgmt
Vendor Analysis

Python Installation on Windows

RV College of Engineering

Installing Python on Windows

Go, change the world



```
# Python 3: List comprehensions
>>> fruits = ['Banana', 'Apple', 'Lime']
>>> loud_fruits = [fruit.upper() for fruit in
fruits]
>>> print(loud_fruits)
['BANANA', 'APPLE', 'LIME']

# List and the enumerate function
>>> list(enumerate(fruits))
[(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]
```

Compound Data Types

Lists (known as arrays in other languages) are one of the compound data types that Python understands. Lists can be indexed, sliced and manipulated with other built-in functions. More about lists in Python 3

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Step 1: Go to **www.python.org** (official website)

Step 2: Go to Download Section and click on Windows

Step 3: Select Version of Python to Install

Step 4: Download Python Executable Installer

Step 5: Run Executable Installer

Step 6: Verify Python Was Installed On Windows

Step 7: Add Python Path to Environment Variables (Optional)

Installation Of Python in Linux

- ✓ Step 1: Download python-3.x version from the Python.org
- ✓ Step 2: Extract .tar.gz using the command tar xvf Python-3.x.tar.gz
  - ✓ (goto that directory using cd dir_name)
- ✓ Step 3: $./configure
- ✓ Step 4: $sudo make
- ✓ Step 5: $sudo apt-get install libssl-dev openssl
- ✓ Step 6: $sudo make install

My First Python Program

Hello World Program

```
// Simple C Program to Display "Hello World"

#include <stdio.h>
int main()
{
        printf("Hello, World!");
        return 0;
}
```

Filename: Hello.c

```
// Simple C++ Program to Display "Hello World"

#include<iostream>
 using namespace std;
 int main()
{
   cout<<"Hello World";
   return 0;
}
```

Filename: Hello.cpp

```
/* Simple Java Program to Display "Hello World"  */

 public class Hello
{
   public static void main(String[] args)
   {
         System.out.println("Hello, World");
   }
}
```

Filename: Hello.java

Once you're inside the Python interpreter, type in commands at will.

Examples:

```
>>> print (“Hello world”)
Hello world
# Relevant output is displayed on subsequent lines without the >>> symbol
>>> x = [0,1,2]
# Quantities stored in memory are not displayed by default
>>> x
# If a quantity is stored in memory, typing its name will display it
[0,1,2]
>>> 2+3
5
```

Executing Python in Linux

Python scripts can be written in text files with the suffix .py. The scripts can be read into the interpreter in several ways:

Examples:

$ python script.py

# This will simply execute the script and return to the terminal afterwards

$ python -i script.py

# The -i flag keeps the interpreter open after the script is finished running

$ python

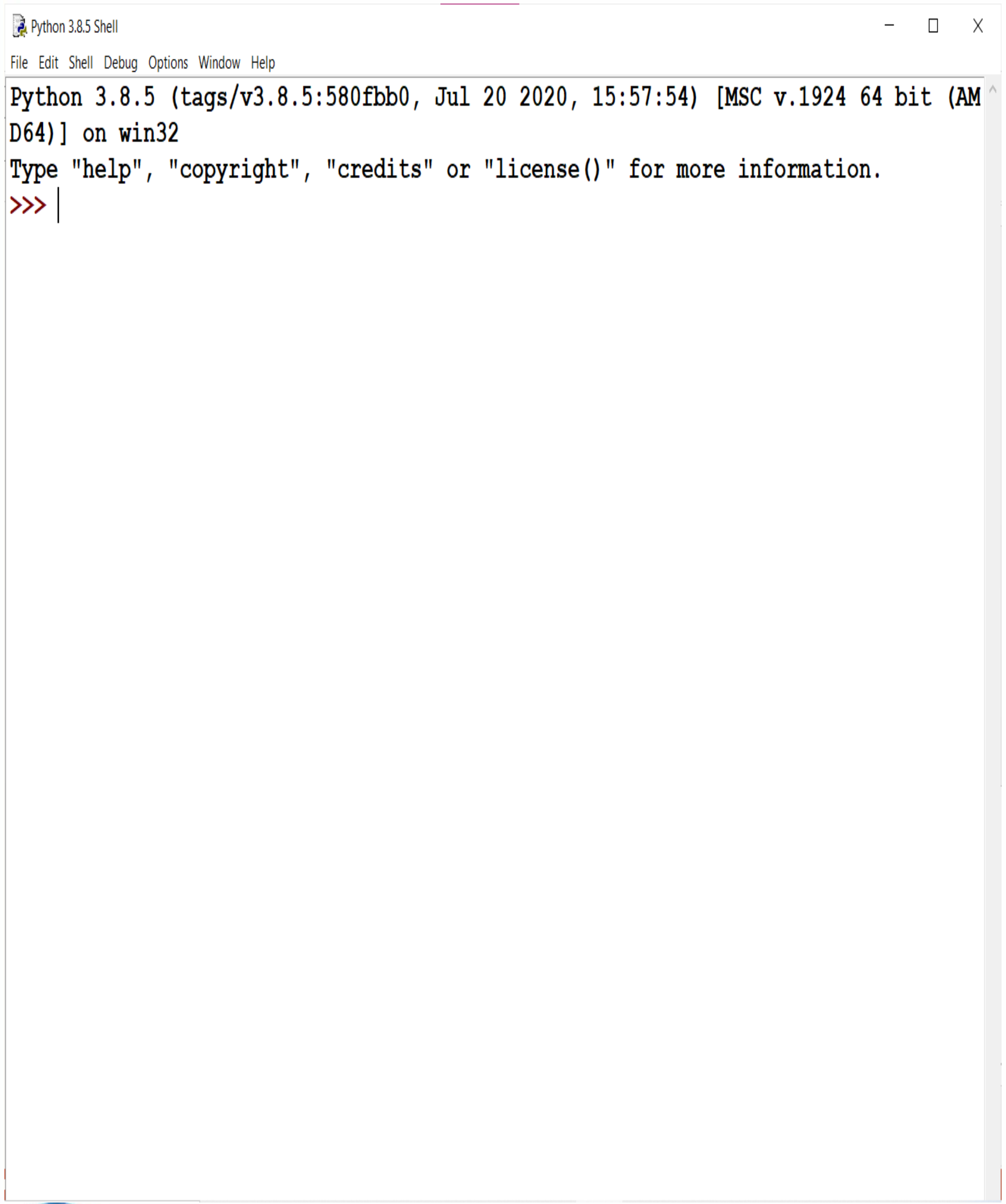
>>> **execfile('script.py')**

# The execfile command reads in scripts and executes them immediately, as though they had been typed into the interpreter directly

$ python

>>> **import script** # DO NOT add the .py suffix. Script is a *module* here

# The import command runs the script, displays any unstored outputs, and creates a lower level (or context) within the program. More on contexts later.



```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
```

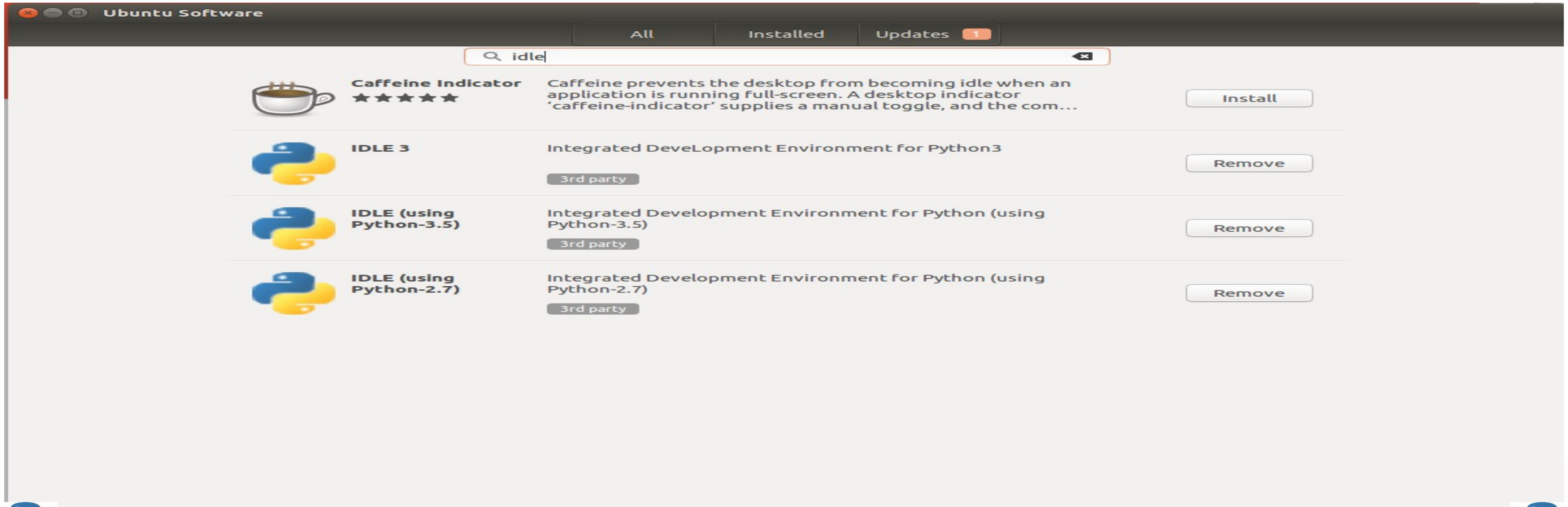
IDLE

- **IDLE** stands for **Integrated Development and Learning Environment Tool for Python**
- **IDLE comes as a Default Implementation along with python**
- **Coded in 100% pure Python**
- **Cross-platform: works mostly the same on Windows, Unix, and macOS**
- **Python IDLE comes with two window**
  - **Shell Window (interactive interpreter)**
  - **Multi-Window with text editor**
- **Colouring of code (Input, Output and Error Messages)**

Installing IDLE

- **Python Version 2**
  - **sudo apt-get update**
  - **sudo apt-get install idle**

- **Python Version 3**
  - **sudo apt-get update**
  - **sudo apt-get install idle3**



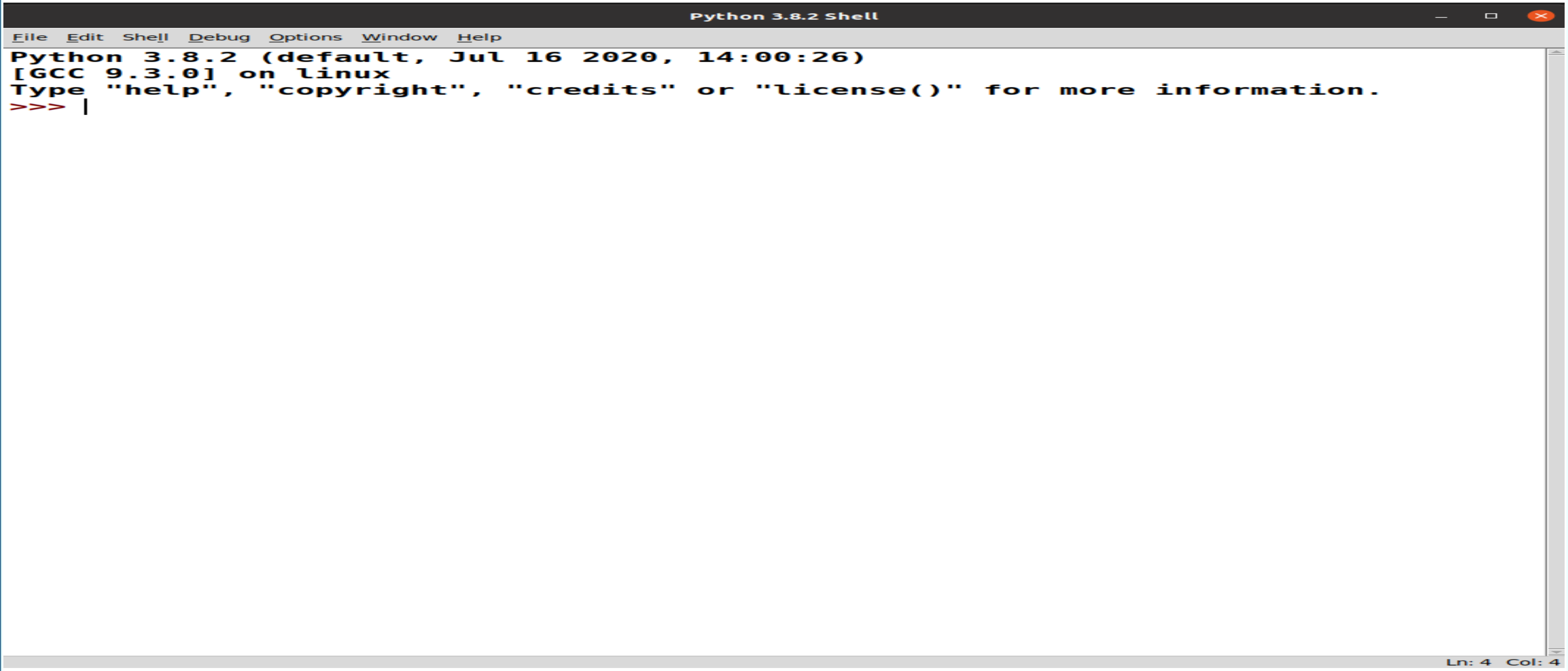
Caffeine Indicator ★★★★★	Caffeine prevents the desktop from becoming idle when an application is running full-screen. A desktop indicator 'caffeine-indicator' supplies a manual toggle, and the com...	Install
IDLE 3	Integrated DeveLopment Environment for Python3 3rd party	Remove
IDLE (using Python-3.5)	Integrated Development Environment for Python (using Python-3.5) 3rd party	Remove
IDLE (using Python-2.7)	Integrated Development Environment for Python (using Python-2.7) 3rd party	Remove

RV College of Engineering

Go, change the world

QUEST SKILLS

Python-IDLE In Ubuntu 20.04



```
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
```

- sudo apt-get update
- sudo apt-get install idle-python3.8

Variable:

- ➢ Variable is a **storage location** paired with an associated symbolic name, which **contains some known or unknown quantity of information** referred to as a value
- ➢ Variables are nothing but **Reserved Memory Locations** to store **values**.
- ➢ Python considers values as **object**

Usage:

- No Declaration,
- Create variable by assigning value and
- Use that variable later in the program

Assignment statement:

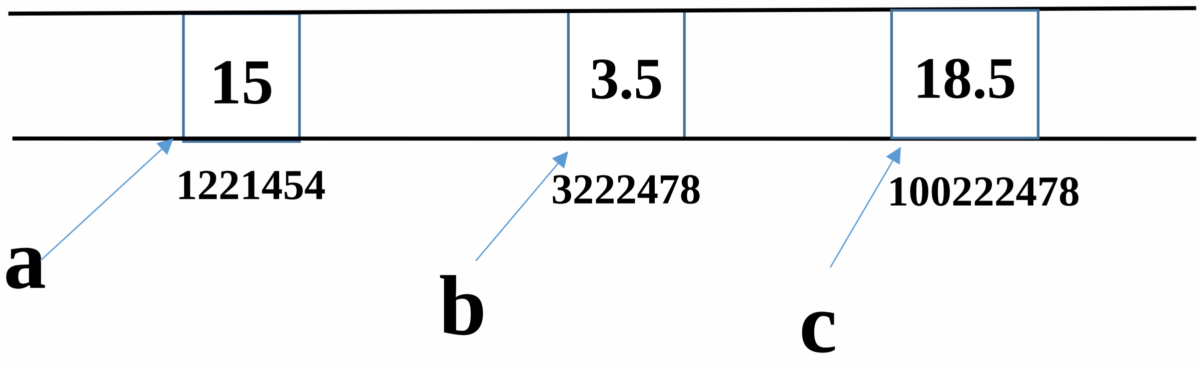
- Stores a value into a variable.

Syntax:

- ***name** = **value***

Examples:

- a = 15
- b = 3.5
- c = a+b



Addition of Two Numbers

```
// Simple C Program to Display "Addition of
Two Numbers"

#include <stdio.h>
int main()
{
            int a = 5;
            int b = 6;
      int sum;
      sum = a+b;
      printf("The addition is %d!", 
sum);
      return 0;
}

Filename: Hello.c
```

```
// Simple C++ Program to Display "Addition
of
//                   Two Numbers"

#include<iostream>
 using namespace std;
 int main()
{
         int a = 5;
                 int b = 6;
         int sum;
         sum = a+b;
         cout<<"Addition is " <<
sum;
         return 0;
}

Filename: Hello.cpp
```

```
/* Simple Java Program to Display "Addition of
Two Numbers"  */

 public class Hello
{
   public static void main(String[]
args)
   {
          int a = 5;
          int b = 6;
          int sum ;
          sum = a+b;

System.out.println("Addition is
"+sum);
    }
}

Filename: Hello.java
```

int a = 5;

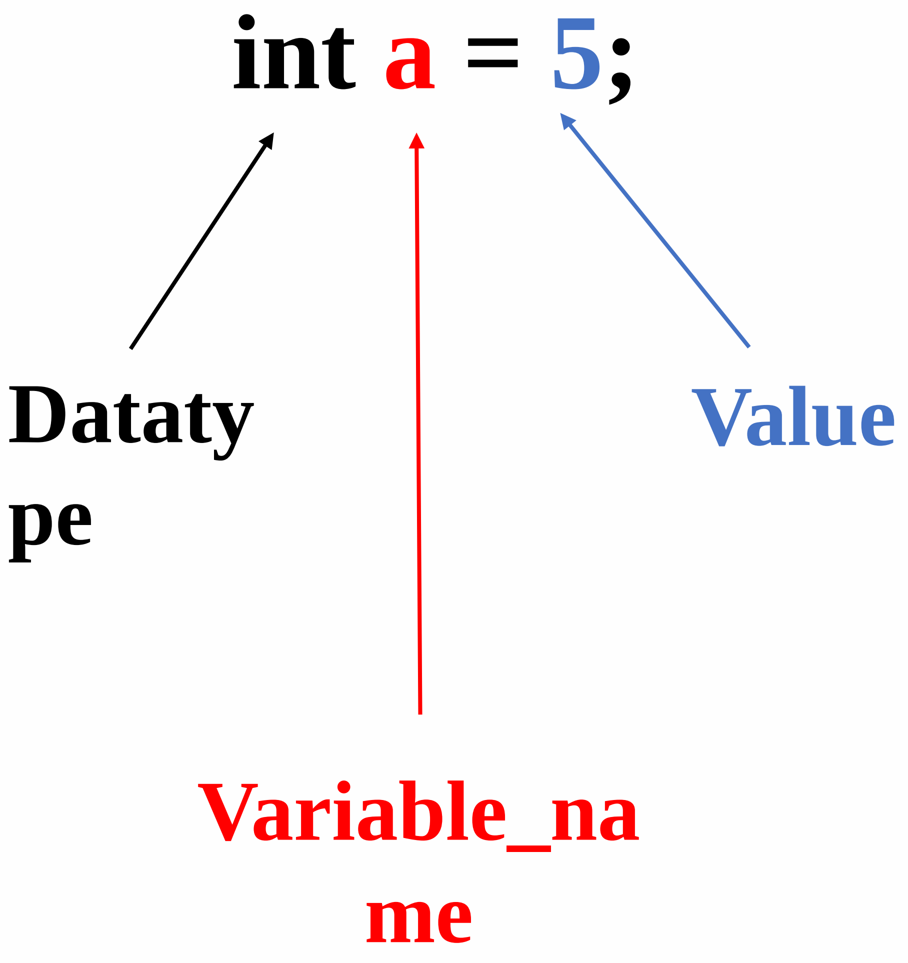
// Simple C Program to Display “Addition of Two Numbers"

```
#include <stdio.h>
int main()
{
            int a = 5;
            int b = 6;
      int sum;
      sum = a+b;
      printf(“The addition is %d!“, sum);
      return 0;
}
```

Filename: Hello.c

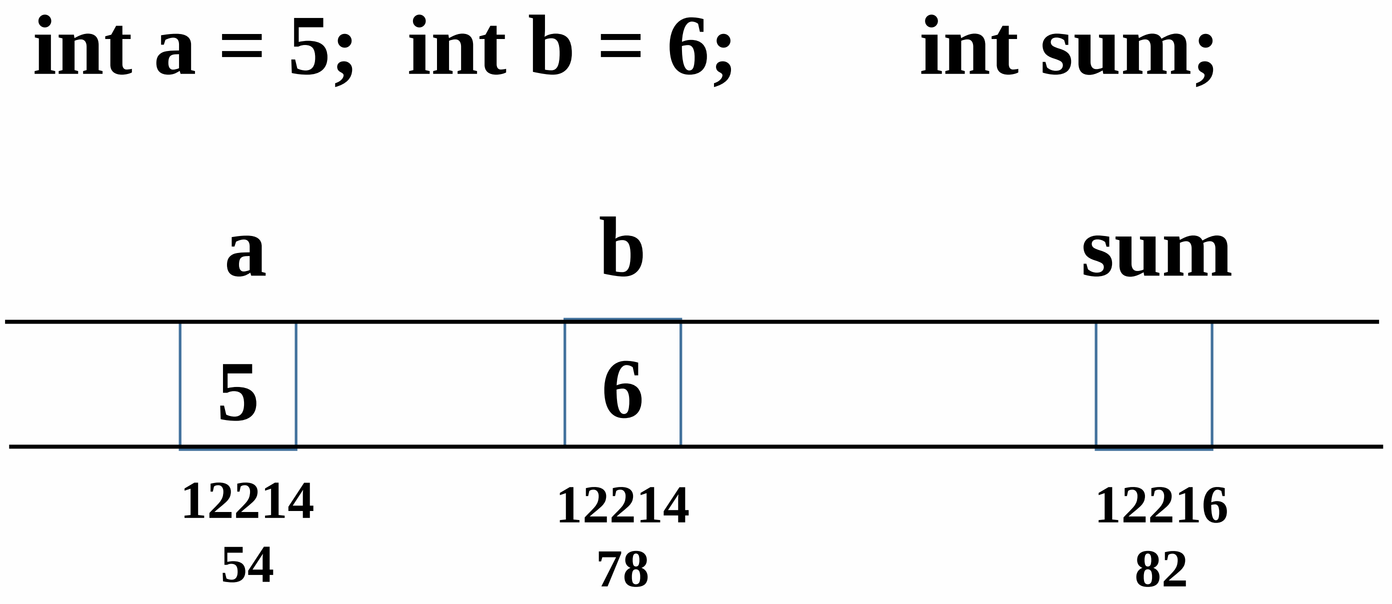
```
// Simple C Program to Display "Addition of
Two Numbers"

#include <stdio.h>
int main()
{
                int a = 5;
                int b = 6;
       int sum;
       sum = a+b;
       printf("The addition is %d!", 
sum);
       return 0;
}
```

Filename: Hello.c



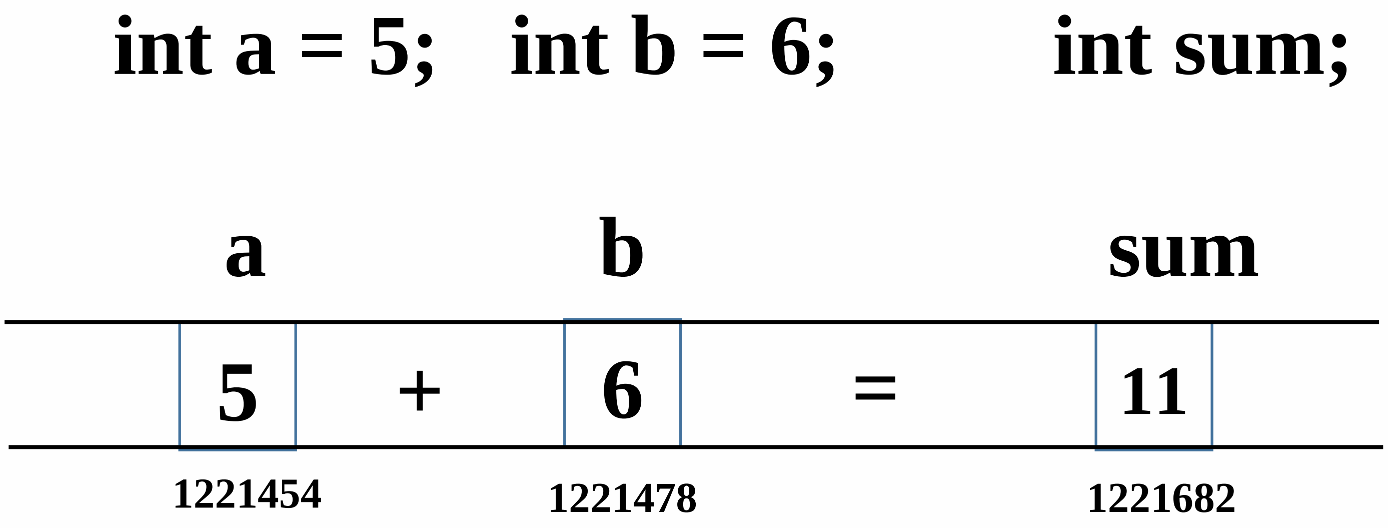
```
#include <stdio.h>
int main()
{
            int a = 5;
            int b = 6;
      int sum;
      sum = a+b;
      printf("The addition is %d!", sum);
      return 0;
}
```

Filename: Hello.c



```
// Simple C Program to Display "Addition of
Two Numbers"

#include <stdio.h>
int main()
{
            int a = 5;
            int b = 6;
      int sum;
      sum = a+b;
      printf("The addition is %d!", sum);
      return 0;
}
```

Filename: Hello.c



Rules:

[illegible]

1

2

3

4

5

6

7

Multiple Assignment of Values

Example 1

```
>>> a = b = c = 1
```

Example 2

```
>>> a, b = 2 , 3
```

Example 3

```
>>> a, b, c = 2, 3.5, “Bangalore”
```

Example 4

```
>>> a, b = b, a
```

Python input()

input():

- **input()** reads a value from user input.
- **input()** is a **built-in Python function** which prompts user to enter some text input.
- If we call input(), the program will **stop at that point and waits** until user inputs some information.
- Program will **resume** once the user presses ENTER key
- input() always **read the string data**

Syntax:

- input([Optional])

Example:

```
age = input("How old are you? ")
print ("Your age is", age)
print ("You have", 65 - age, "years until retirement“)
```

Command Line Arguments:

- Python also **supports command line arguments** which we can pass during run time
- **sys module** is used for parsing the command line arguments

Operators

- Python **operator is a symbol** that performs an operation on one or more operands.
- An **operand is a variable** or a value on which we perform the operation.
- The standard mathematical operators that you are familiar with work the same way in Python as in most other languages.
- Python Operator falls into **7 categories**
- Python language supports the following types of operators.
  - Arithmetic Operators (+ - * / // % **)
  - Comparison (Relational) Operators (> < == <= >= !=)
  - Assignment Operators (= += -= /= *= //= **=)
  - Logical Operators (and or not)
  - Bitwise Operators (& | ^ << >> ~)
  - Membership Operators (in not in)
  - Identity Operators (is is not)

Expressions and Statements

An expression is a combination of values, variables, and operators.

```
 >>> 1+1
   2
>>> x = 5
>>> x+5
10
```

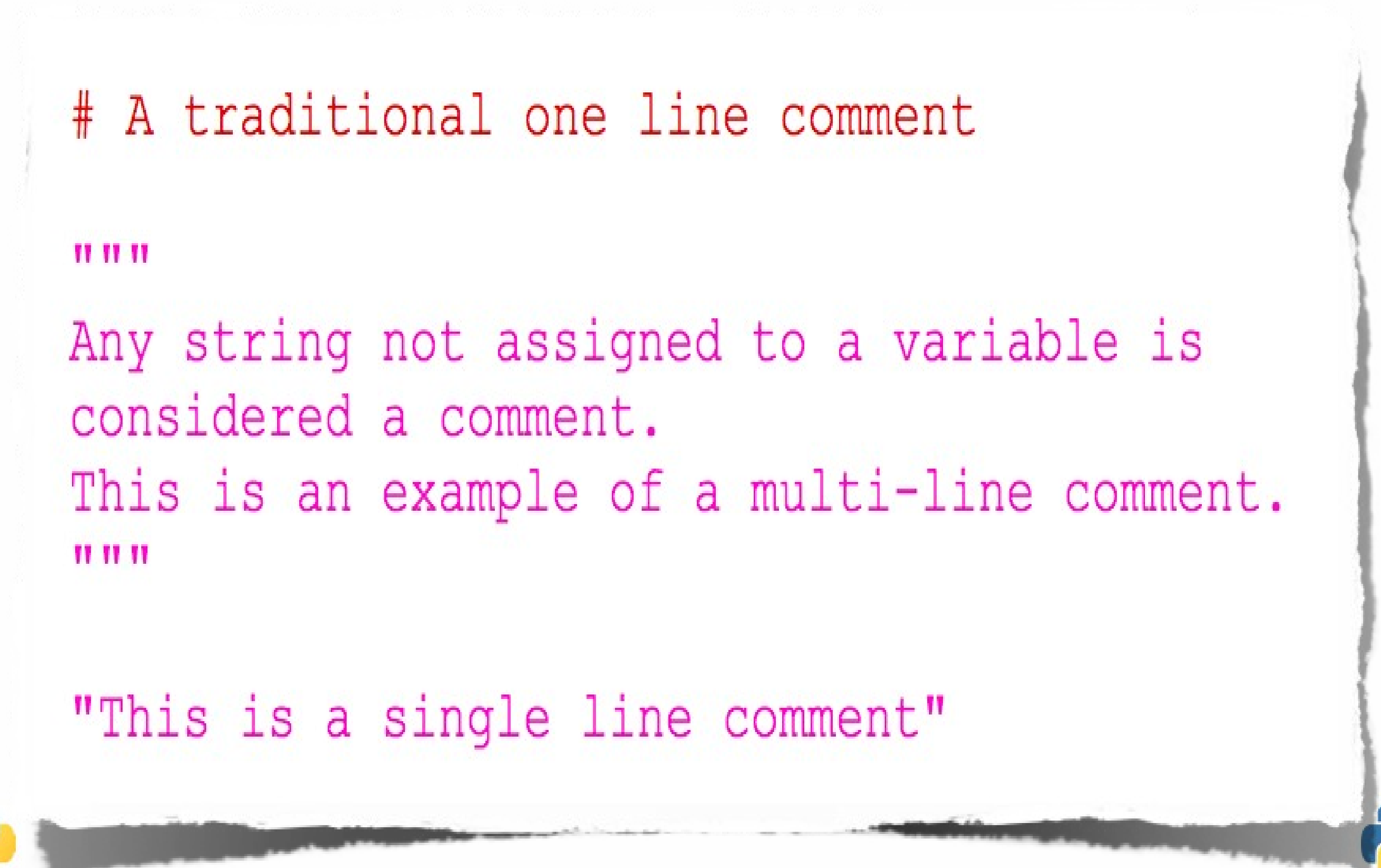
```
Composition
>>> a, b = 10, 5.3
>>>c = a*60+5.3
```

Order of operations: PEMDAS

- ✓**P**arentheses
- ✓**E**xponentiation
- ✓**M**ultiplication and **D**ivision
- ✓**A**ddition and **S**ubstraction (Operators with the same precedence are evaluated from left to right)

```
>>>6+4/4*10/5
```

Comments



```
# A traditional one line comment

"""
Any string not assigned to a variable is
considered a comment.
This is an example of a multi-line comment.
"""

"This is a single line comment"
```

Indentation

- Most languages don’t care about indentation
- Most humans do
- We tend to group similar things together

```
/* C Code */

If (foo) {
        If (bar) {
                baz(foo, bar);
}
else {
        qux();
}  }
```

Basic Statements: The If Statement (1)

If statements have the following basic structure:

```
# inside the interpreter              # inside a script
>>> if condition:                       if condition:
...         action                      ..    action
...
>>>
```

Subsequent indented lines are assumed to be part of the if statement. The same is true for most other types of python statements. A statement typed into an interpreter ends once an empty line is entered, and a statement in a script ends once an unindented line appears. The same is true for defining functions.

If statements can be combined with else if (elif) and else statements as follows:

```
if condition1:     # if condition1 is true, execute action1
      action1
elif condition2:   # if condition1 is not true, but condition2 is, execute
      action2      # action2
else:              # if neither condition1 nor condition2 is true, execute
      action3      # action3
```

Conti…

If statment

Syntax:

```
if expression:
        statement(s)
```

Example:

```
num = 3
if num > 0:
        print(num, "is a positive number.")
print("This is always printed.")
num = -1
if num > 0:
        print(num, "is a positive number.")
print("This is also always printed.")
```

Nested if Statement

Syntax:

```
if expression:
            statement(s)
elif expression:
            statement(s)
else:
            statement(s)
```

Example:

```
num = 75
if num >100:
        print(num, "is a greater than 100")
elif  num < 50:
        print(num, “less than 50”)
else:
        print (num,”less than 50  )
print("This is also always printed.")
```

Basic Statements: The While Statement (1)

While statements have the following basic structure:

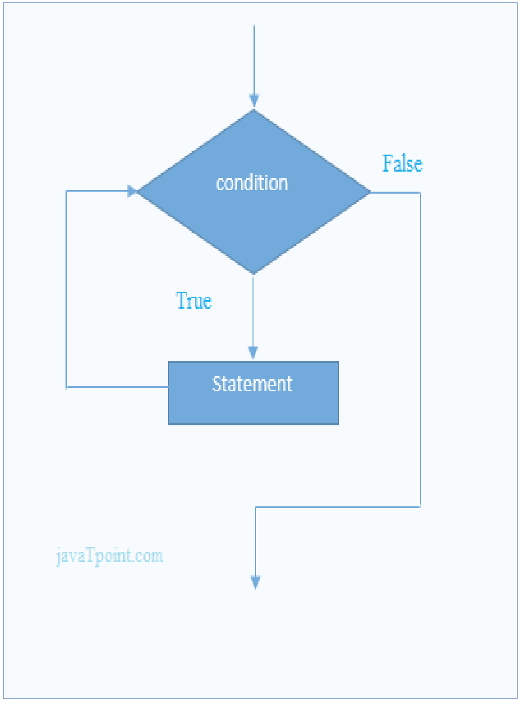
```
# inside the interpreter
>>> while condition:
...                action
...
>>>
```

```
# inside a script
while condition:
                action
```

As long as the condition is true, the while statement will execute the action

Example:

```
>>> x = 1
>>> while x < 4:            # as long as x < 4...
...           print x**2        # print the square of x
...           x = x+1         # increment x by +1
...
1                           # only the squares of 1, 2, and 3 are printed, because
4                           # once x = 4, the condition is false
9
>>>
```



Continuation (\)

- Python statements are in general, delimited by NEWLINEs, meaning one statement per line.
- Single statements can be broken up into multiple lines by use of the backslash.
- The backslash symbol (\ ) can be placed before a NEWLINE to continue the current statement onto the next line.

Example:

```
# check conditions
If  (weather_is_hot == 1)  and  \
    (shark_warnings == 0):
        send_goto_beach_mesg_to_pager()
```

Multiple Statements Groups as Suites (:)

- Groups of individual statements making up a single code block are called "**suites**" in Python.
- Compound or complex statements, such as **if , while , def , and class** , are those that require a header line and a suite.
- We will refer to the combination of a header line and a suite as a clause.
- **Examples:**
  - If condition :

    ```
    pass
    ```
  - While condition:

    ```
    pass
    ```
  - def name() :

    ```
    pass
    ```

Multiple Statements on a Single Line (;)

- The semicolon (;) allows multiple statements on a single line given that neither statement starts a new code block.

- **Example**:

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

Variable Assignment

- **Assignment Operator (=)**
  - **Example**:

```
A = 10
B = "RVCE"
C =  A = A + 1
C =  (A = A + 1) <--- Error (Its Not Expression
```

- **Augmented Assignment**

  Beginning in Python 2.0, the equal sign can be combined with an arithmetic operation and the resulting value reassigned to the existing variable. Known as augmented assignment, statements

  - **Example:**

```
x = x + 1 ==> x += 1
+=   -=  <<=   >>=   *=   &=   /=    ^= |=   %=   **=
```

Python Identifiers

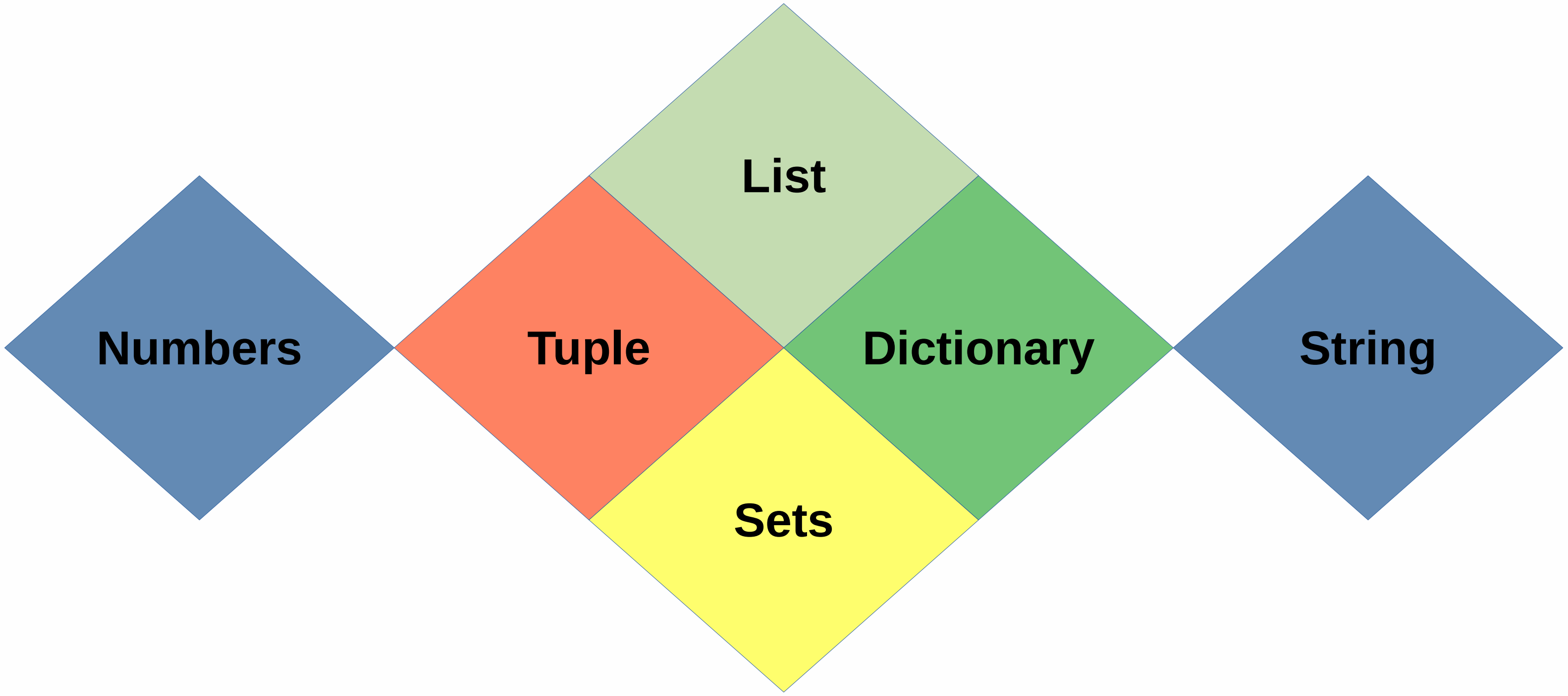
- A Python identifier is a name used to identify a
  - Variables,
  - Functions,
  - Class,
  - Modules or
  - Other object.
- An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers.
- Python is a case sensitive programming language.
- Thus, **RVCE** and **rvce** are two different identifiers in Python.
  - First character must be a letter or underscore (_)
  - Any additional characters can be alphanumeric or underscore
  - Case-sensitive

Data Types

Data Types

- Data type determines the set of values that a data item can take and the operations that can be performed on the item.

- Python language provides Six basic data types.



1. Numbers

- A particular kind of data item, as defined by the **values it can take, the programming language used, or the operations that can be performed on it**
- Number data types store numeric values.
- They are immutable data types
- **Example:**

```
Var = 1
Var = 2
```

- Python supports four different numerical types
  - int
  - long
  - float
  - complex

Cont… (numbers)

int	long	float	complex
10	519266L	0.0	5.24j
100	-0x313L	3.5	3.14j
-786	01254L	-6.9	9.322ej
0	0xDEFABCCEDL	54+e52	0.875j

Number Type Conversion:

- Type **int(x)** :convert x to a plain integer.
- Type **long(x)** :convert x to a long integer.
- Type **float(x)** :convert x to a floating-point number.
- Type **complex(x)** :convert x to a complex number with real part x and imaginary part zero.
- Type **complex(x, y) :**convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions

Mathematical Functions:

abs(x), random.random(), log(x), choice(seq), etc……..

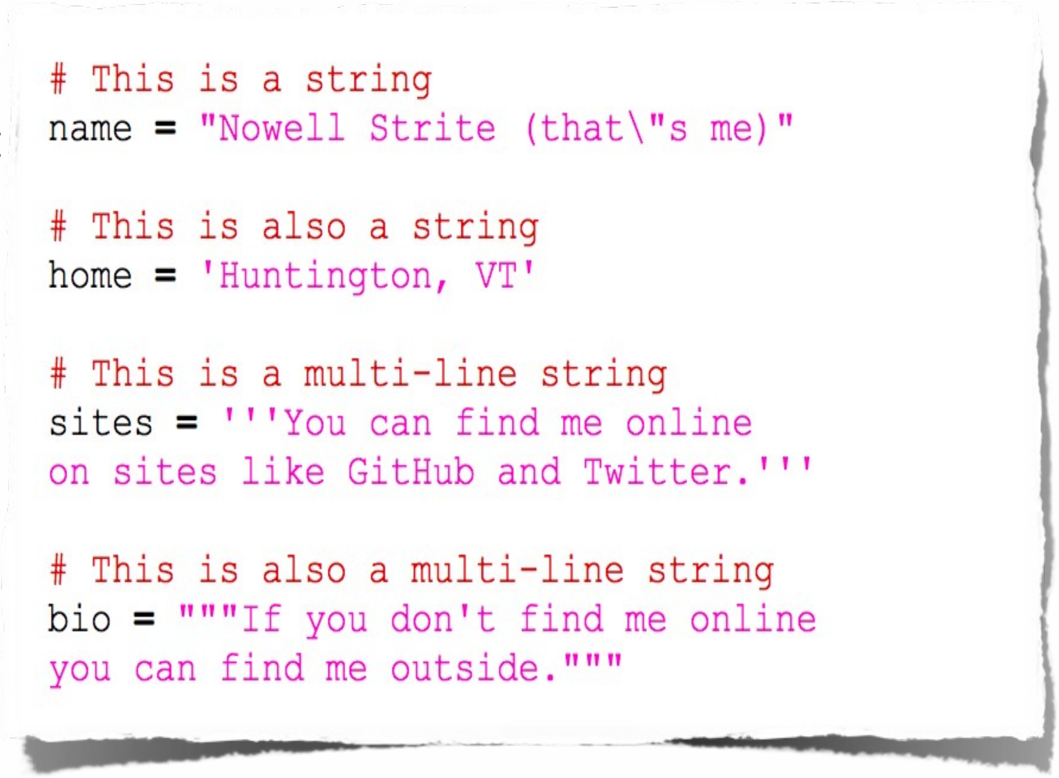
2. String

- Strings are enclosed in **single or double quotation marks**
- Double quotation marks allow the user to extend strings over multiple lines without backslashes, which usually signal the continuation of an expression
- Examples: **'abc', "ABC", """ABC"""**

Concatenation and repetition

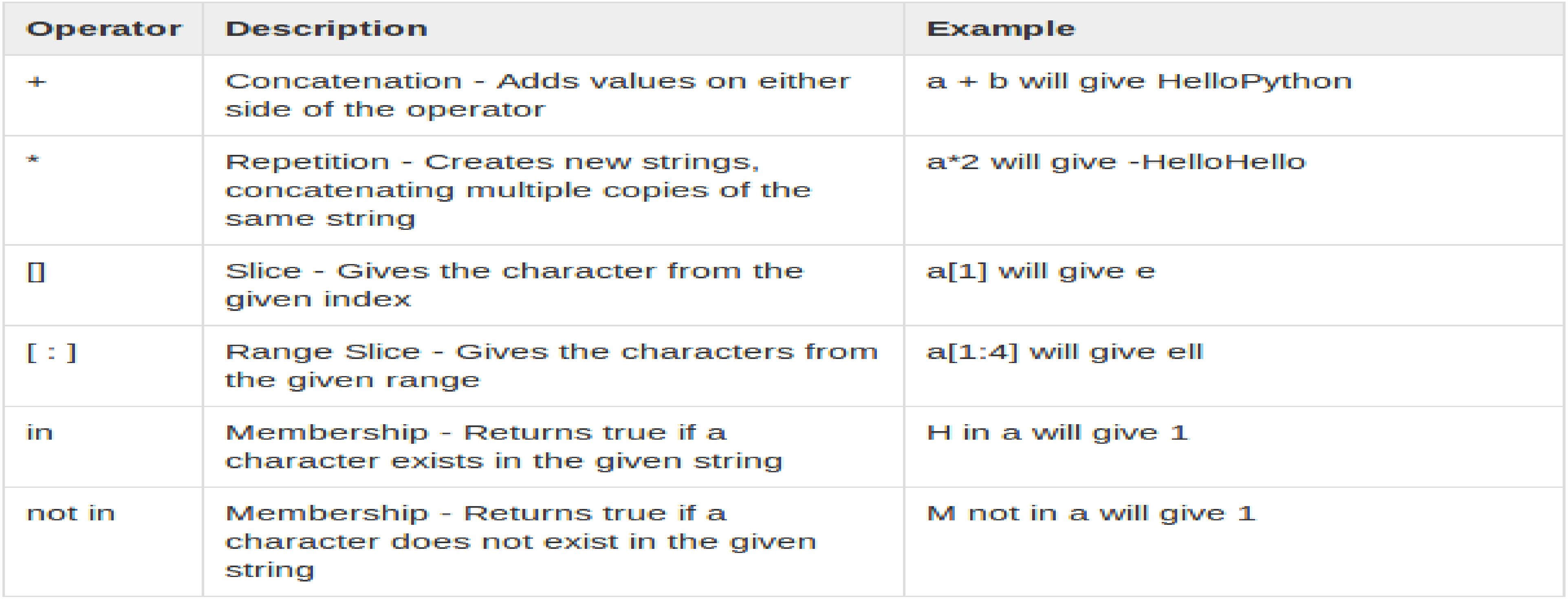
Strings are concatenated with the + sign:
>>> 'abc'+'def'
'abcdef'

Strings are repeated with the * sign:
>>> 'abc'*3
'abcabcabc'



```
# This is a string
name = "Nowell Strite (that\"s me)"

# This is also a string
home = 'Huntington, VT'

# This is a multi-line string
sites = '''You can find me online
on sites like GitHub and Twitter.'''

# This is also a multi-line string
bio = """If you don't find me online
you can find me outside."""
```

String Operatons

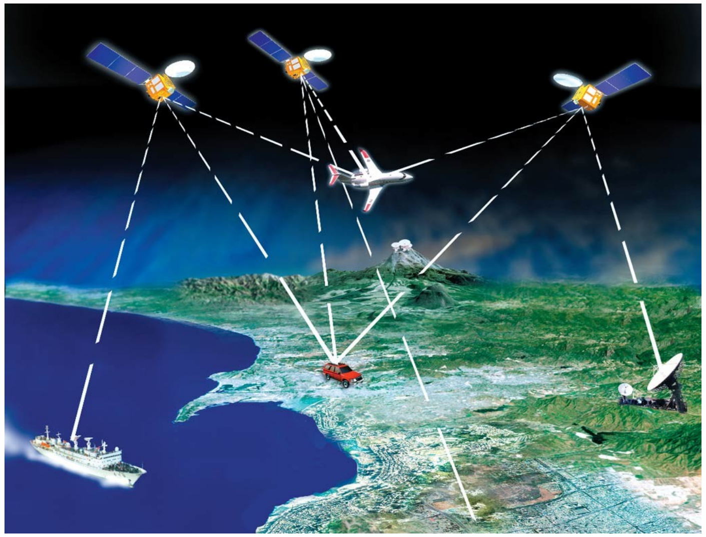


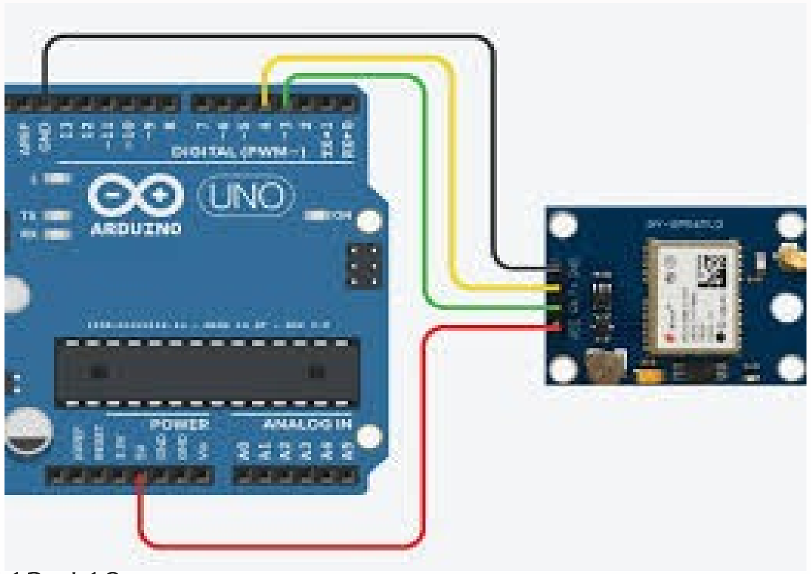
Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1

```
>>>print ("IoT workshop jointly organised by %s and %s !" % ('RVCE', ‘CISCO’))
```

Output:

```
IoT workshop jointly organised by RVCE and CISCO !
```





- $GPRMC,235316.000,A,4003.9040,N,10512.5792,W,0.09,144.75,141112,,*19

```
>>>s = “GPRMC,235316.000,A,4003.9040,N,10512.5792,W,0.09,144.75,141112,,*19”
>>>a, b = s[19:28], s[31:41]
>>>a, b
```

output:

```
('4003.9040', '10512.5792')
```

```
>>>a=float(a), b=float(b)
```

Output:

```
(4003.904, 10512.5792)
```

3. Lists:

Basic properties:

- Lists are contained in square brackets []
- Lists can contain numbers, strings, nested sublists, or nothing

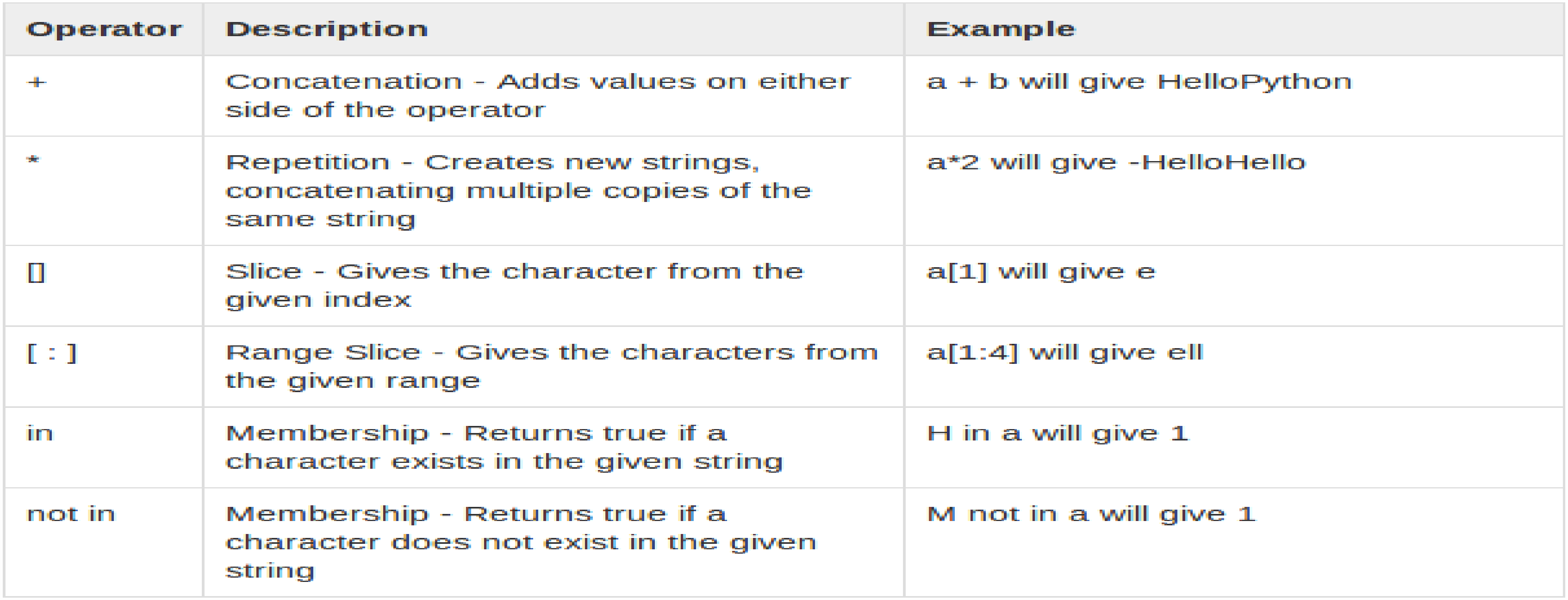
Examples:

- **L1 = [0,1,2,3]**
- **L2 = ['zero', 'one']**
- **L3 = [0,1,[2,3],'three',['four,one']]**
- **L4 = []**

- List indexing works just like string indexing
- Lists are mutable: individual elements can be reassigned in place. Moreover, they can grow and shrink in place
- Example:

```
>>> L1 = [0,1,2,3]
>>> L1[0] = 4
>>> L1[0]
4
```

List Operations



Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1

Some basic operations on lists:

- ✓Indexing: L1[i], L2[i][j]
- ✓Slicing: L3[i:j]
- ✓L1.append(x)
- ✓L1.sort()

4. Tuple:

Basic properties:

- Tuples are contained in parentheses ()
- Tuples can contain numbers, strings, nested sub-tuples, or nothing
- Examples:
  - t1 = (0,1,2,3),
  - t2 = ('zero', 'one'),
  - t3 = (0,1,(2,3),'three',('four,one')),
  - t4 = ()
- As long as you're not nesting tuples, you can omit the parentheses
  Example: t1 = 0,1,2,3 is the same as t1 = (0,1,2,3)
- Tuple indexing works just like string and list indexing
- Tuples are **immutable**: individual elements cannot be reassigned in place.
- **Concatenation:**

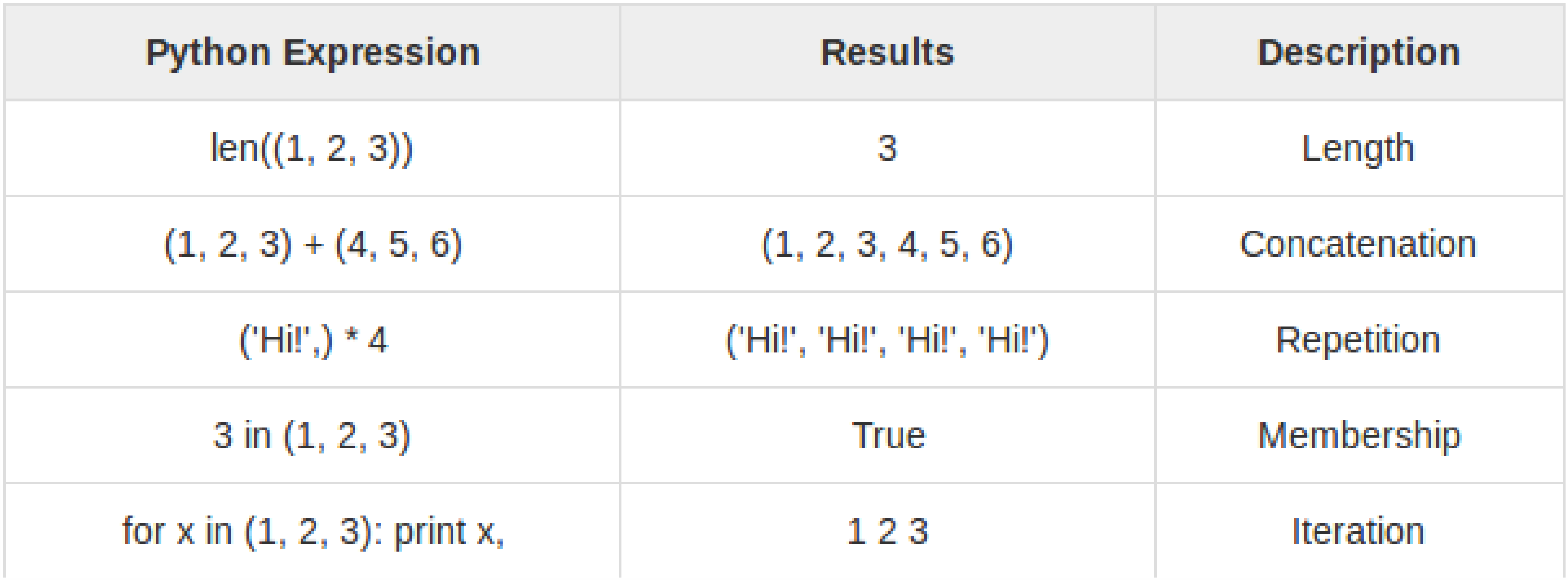
```
>>> t1 = (0,1,2,3); t2 = (4,5,6)
>>> t1+t2
(0,1,2,3,4,5,6)
```

- **Repetition:**

```
>>> t1*2
(0,1,2,3,0,1,2,3)
```

- Length: len(t1) (this also works for lists and strings)

Tuple:



Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

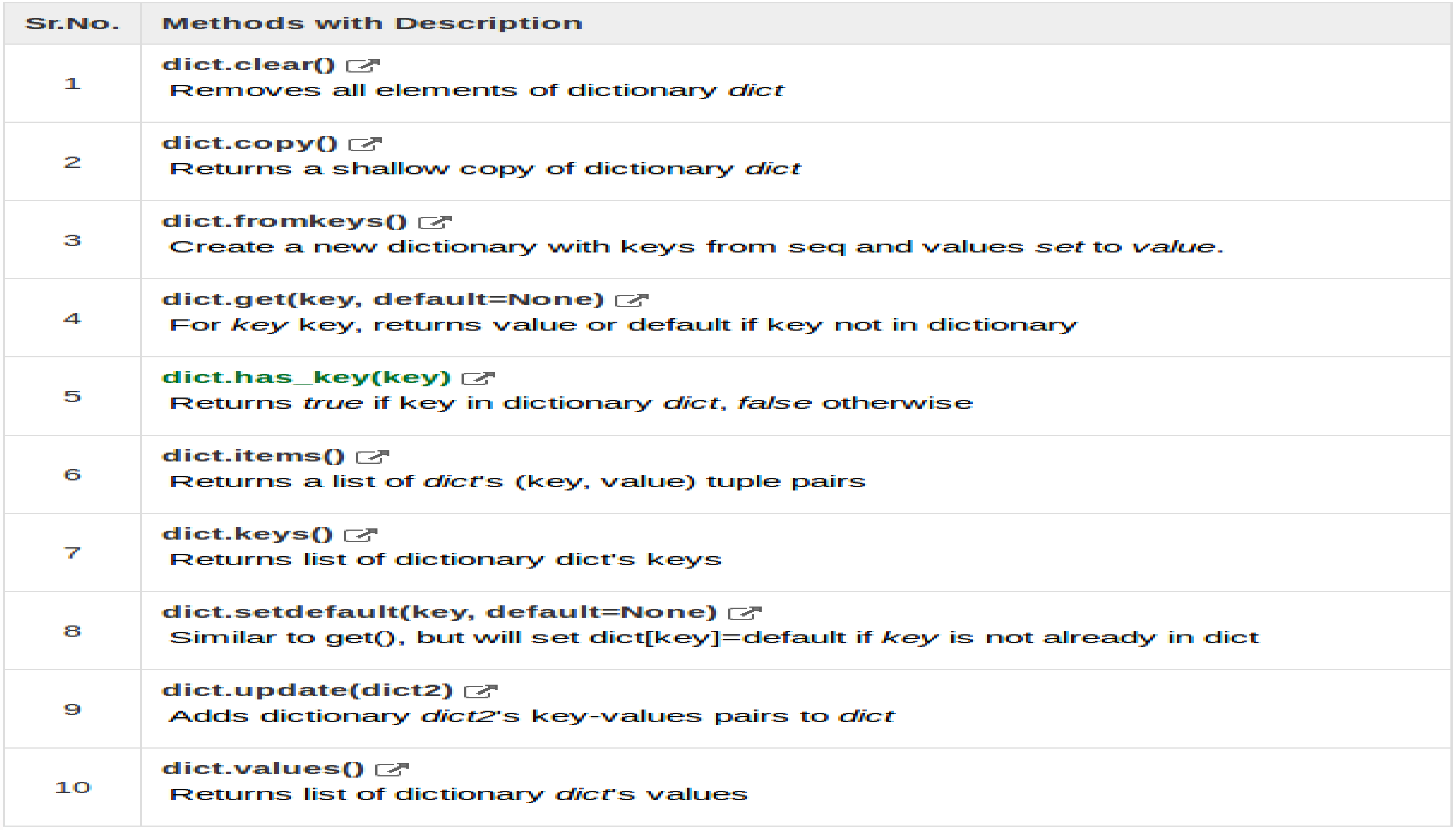
```
tup1 = ('RVCE', 'Workshop', 2018, 2017)
tup2 = (1, 2, 3, 4, 5, 6, 7 )
print "tup1[0]: ", tup1[0]
print "tup2[1:5]: ", tup2[1:5]
```

5. Dictionary:

- Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces.
- Empty dictionary = { }
- Keys are unique within a dictionary while values may not be.
- The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.
- Example:

```
dict = {'College': 'RVCE', 'Workshop': ‘IoT’, 'Place': 'Bangalore', ‘Date’:17}
```

Dictionary Operations:



Sr.No.	Methods with Description
1	**dict.clear()** Removes all elements of dictionary *dict*
2	**dict.copy()** Returns a shallow copy of dictionary *dict*
3	**dict.fromkeys()** Create a new dictionary with keys from seq and values *set* to *value*.
4	**dict.get(key, default=None)** For *key* key, returns value or default if key not in dictionary
5	**dict.has_key(key)** Returns *true* if key in dictionary *dict*, *false* otherwise
6	**dict.items()** Returns a list of *dict*'s (key, value) tuple pairs
7	**dict.keys()** Returns list of dictionary dict's keys
8	**dict.setdefault(key, default=None)** Similar to get(), but will set dict[key]=default if *key* is not already in dict
9	**dict.update(dict2)** Adds dictionary *dict2*'s key-values pairs to *dict*
10	**dict.values()** Returns list of dictionary *dict*'s values

6. Sets:

- ✓ Sets are used to store multiple items in a single variable.
- ✓ A set is a collection which is both unordered and unindexed.
- ✓ Sets are written with curly brackets.
- ✓ An empty set is created as “a = set()”
- ✓ Sets are unordered.
- ✓ Set elements are unique.
- ✓Duplicate elements are not allowed.
- ✓ A set itself may be modified, but the elements contained in the set must be of an immutable type.
- ✓Example:

```
    a = {1, 2, 3, 5, 6, 'RVCE'}
```

For Statment:

Syntax:

```
for item i in set s:
            action on item i
```

# item and set are not statements here; they are merely intended to clarify the relationships between i and s

Example:

```
>>> for i in range(1,7):
     …..    print (i, i**2, i**3, i**4)
     ….
Output:
        1 1 1 1
        2 4 8 16
        3 9 27 81
        4 16 64 256
       5 25 125 625
        6 36 216 1296
>>>
```

Functions

- Set of statements to perform specfic task.
  Built-in fuctions **Our Own functions**

- Built-in functions
  <<function_name>>(<<arg>>)

  abs(), round(value,[No.places]), pow(val1,val2,[val3])

- **round(3.4)** 3 **round(3.6)** 4 **round(5.31243,2)** 5.31

  pow(2,3) 8 **pow(2,3,3)** 2 (8%3)

help(built_in func) id(object) hex(decimal) oct(decimal)
A=9, B=9, id(A), id(B), id(9) output: Same id
Note: Objects with same value refers to same memory

Our Own Functions

- Functions as object
- Provides modularity and reusability
- Syntax

```
def functionname( parameters ):
       "Document string"
        Statements
       return [expression]
```

```
>>> id(print_hello)
139710611025432
>>> type(print_hello)
<class 'function'>
```

```
>>> def print_hello(str1,str2):
            "demo of my own function"
            print(str1,str2)

>>> print_hello("hello","how r u")
hello how r u
```

Parameter Passing

- Pass by Reference by default

```
>>> def argdemo(nlist):
        nlist.append(['r','v','c','e'])
        print("LIST INSIDE",nlist)
        Return

>>> nlist=[1,2,3]
>>> nlist
[1, 2, 3]
>>> argdemo(nlist)
LIST INSIDE [1, 2, 3, ['r', 'v', 'c', 'e']]
>>> nlist
[1, 2, 3, ['r', 'v', 'c', 'e']]
```

Types of parameter passing

- **Function can take four different types of arguments**

  Required arguments

  Keyword arguments

  Default argumets

  Variable length argumets

Required Arguments

Arguments must be passed in correct number and order during function call

def requiredArg (str,num):

requiredArg ("Hello",12) \\ function call

```
def lsearch(a,n):
    p=-1
    for i in range(0,len(a)):
        if a[i]==n:
            p=i
            break
    if p>-1:
        print("Element found at",p+1)
    else:
        print("Element not found")
    return
```

```
>>> a=[20,10,100,4,6]
>>> lsearch(a,4)
Element found at 4
>>> lsearch(a,80)
Element not found
```

Keyword Arguments

Arguments can be passed in any order during function call

keywords provided to match the values with parameters

```
def concat(b,a):
    print ("Message from the function\n")
    c = a + b
    print (c)


concat(a="MCA  ",b= "2")
```

Default Arguments

Arguments will take the default value if no argument value is passed during function call

```
def defaultargument(a, b = 24):
    print ("Message from the function Default Arguments\n")
    print ("Name ==> ", a)
    print ("Age ===> ", b)


defaultargument(a="Miki",b= "26")
defaultargument(a="Miki")
```

Variable number of Arguments

Variable number of arguments can be passed based on requirement during function call

```
def printinfo( arg1, *vartuple ):
    "This prints a variable passed arguments"
    print ("Output is: ")
    print (arg1)
    for var2 in vartuple:
        print (var2, vartuple)
    return

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
printinfo( 70, 60, 50, 40, 30, 20, 10, -1 )
```

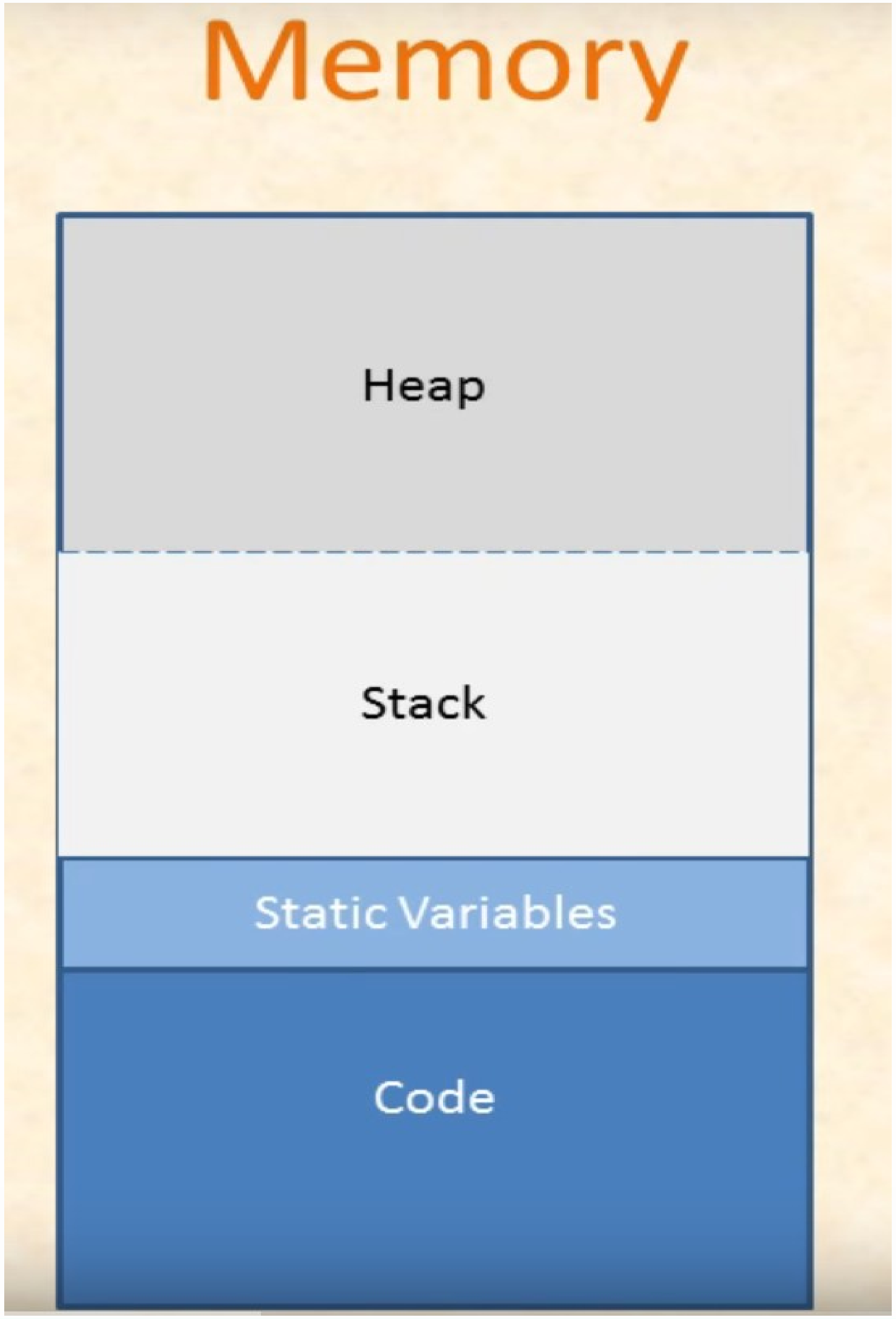
Recursion

Defination:

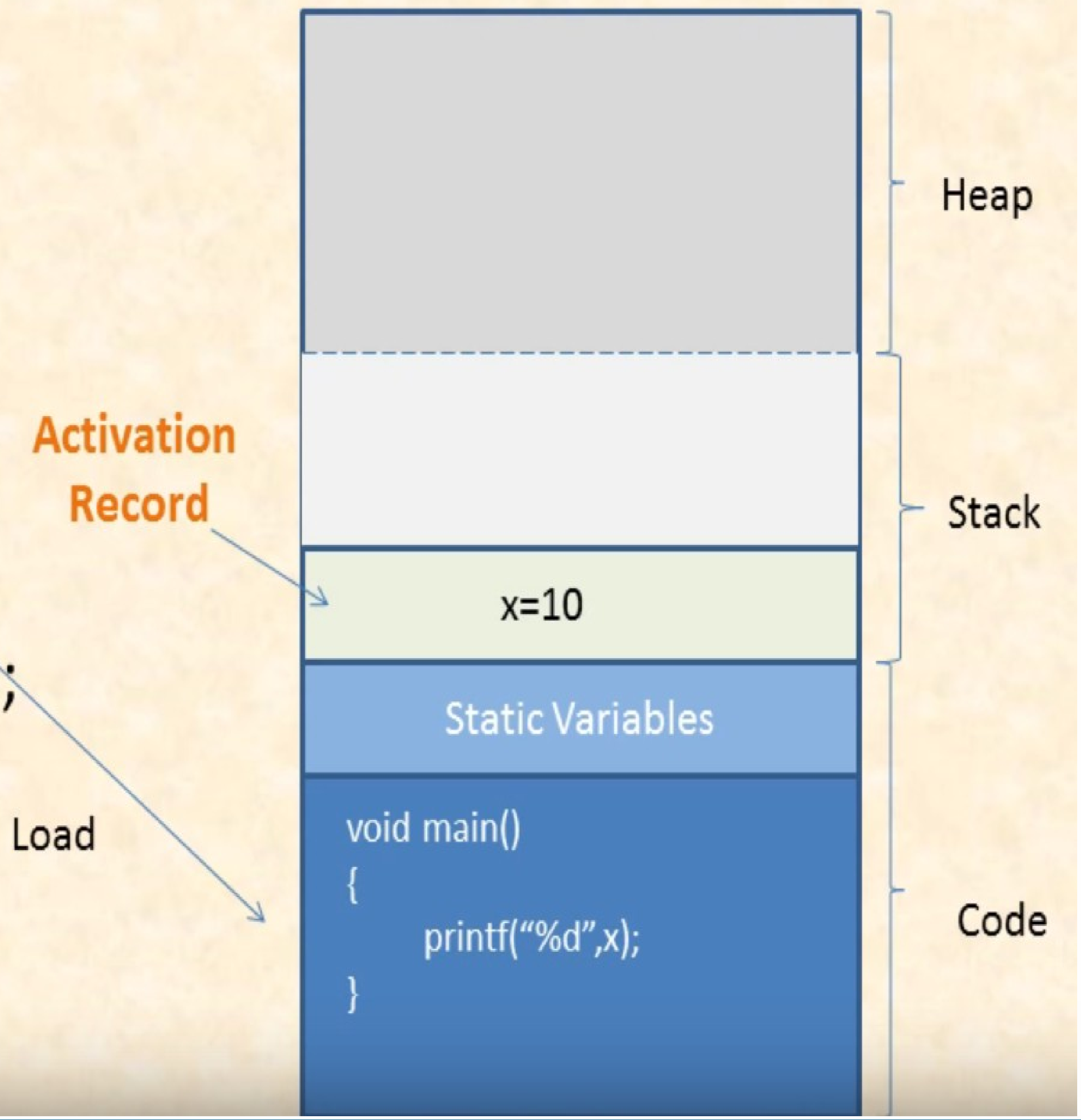
- Recursion in computer science is a method where the solution to a problem depends **on solutions to smaller instances of the same problem**.

- In programming terms, **the process of function calling itself is called recursion**. Of cource, it must be possible for the recursive function to sometimes be completed without the recursive call.

- Example:
  Fact(5) = 5 * Fact(4)
  Fact(5) = 5 * 4 * Fact(3)
  Fact(5) = 5 * 4 * 3 * Fact(2)
  Fact(5) = 5 * 4 * 3 * 2 * Fact(1)
  Fact(5) = 5 * 4 * 3 * 2 * 1
  Fact(5) = 5 * 4 * 3 * 2
  Fact(5) = 5 * 4 * 6
  Fact(5) = 5 * 24
  Fact(5) = 120
-


Memory
Heap
Stack
Static Variables
Code

```
void main()
{
        int x=10;
        printf("%d",x);
}
```



Lamda

- Python allows one to create anonymous functions using the **lambda** keyword
- Lamda functions are called anonymous because they are not declared in the standard manner by using the def keyword.
- You can use the lambda keyword to create small anonymous functions.
- Lambda forms can take any number of arguments but return just one value in the form of an expression.
- They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
- **Syntax**
- The syntax of lambda functions contains only a single statement, which is as follows −

```
lambda [arg1 [,arg2,.....argn]]:expression
```

Filter()

- The **filter()** method constructs an iterator from elements of an iterable for which a function returns true.
- **filter()** method filters the given iterable with the help of a function that tests each element in the iterable to be true or not.
- Syntax:

  filter(function, iterable)

- filter() method takes two parameters:
  - **function** -
    - function that tests if elements of an iterable return true or false.
    - If None, the function defaults to Identity function - which returns false if any elements are false
  - **iterable** -
    - iterable which is to be filtered, could be sets, lists, tuples, or containers of any iterators

map()

- The map() function executes a specified function for each item in an iterable.
- The item is sent to the function as a parameter.
- map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)
- **Syntax:**

  map(function, iterable)
- map() method takes two parameters:
  - **function** -
    - It is a function to which map passes each element of given iterable.
  - **iterable** -
    - It is a iterable which is to be mapped.

Procedural Language .vs. Object Oriented Programming Langauage

- Procedural language is **based on functions** but object oriented language is **based on real world objects**.
- Procedural language gives importance **on the sequence of function execution** but object oriented language gives importance **on states and behaviors of the objects**.
- Procedural language **exposes the data to the entire program** but object oriented language **encapsulates the data**.
- Procedural language follows **top down programming paradigm** but object oriented language follows **bottom up programming paradigm**.
- Procedural language is **complex in nature** so it is difficult to modify, extend and maintain but object oriented language is **less complex in nature** so it is easier to modify, extend and maintain.
- Procedural language provides **less scope of code reuse** but object oriented language provides **more scope of code reuse**.

Class members (Data member and Member function)

Data Members:

- Variables defined in a class are called data memebers.
- There are types of fields Class Variables and Instance Variables
  - **Class variable:**
    - The variables accessed by all the objects(instances) of a class are called calss varibales.
    - There is only copy of the class variable and when any one object makes a change to a class variable, the change is reflected in all the other instances as well.
    - Class variables can be accessed using the syntax
      - Class_name.Class variable
      - Object_name. Class variable
  - **Object/instance Variable:**
    - Variables owned by each individual object (instance of a class) are called object variables.
    - Each object has its own copy of the field i.e. they are not shared and are not related in any way to the field by the same name in a different instance of the same class.
- **Note:** ***if user changes the value of class variable using class name, then it reflects to all objects if user changes the value of class variable using object name, then it reflects only to that objects***

Member Function(method)

Member function(method):

- A method is a function that is defined inside a class.
- Methods are members of classes.
- A method is a function that is available for a given object.
- There are different types of methods like
  - Class methods,
  - Instance methods,
  - Service methods, and
  - Support methods.

Class method:

- A "class method" is a method where the initial parameter is not an instance object(self), but the class object , which by convention is called cls .
- This is implemented with the @classmethod decorator.
- The class method does not require object to invoke it.

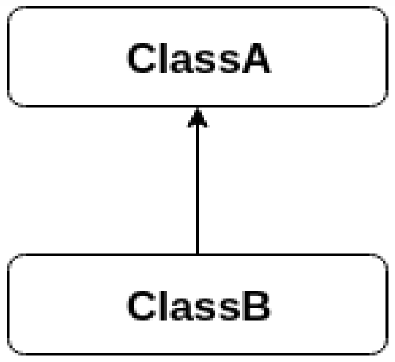
Instance Method:

- An "Instance method" is a method where the initial parameter is an instance of object(self) and requires no decorator.
- These are the most common methods used.

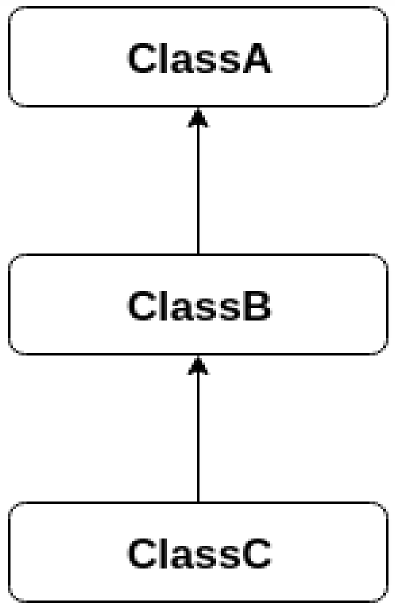
Inheritance

Inheritance

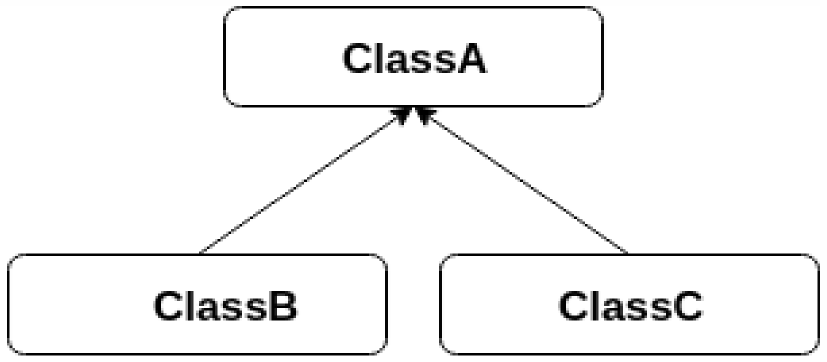
- Inheritance is the ability to define a new class that is a modified version of an existing class.
- The primary advantage of this feature is that you can add new methods to a class without modifying the existing class.
- The existing class is called as Super Class/Parent class.
- A Subclass, "derived class", their class, or child class is a derivative class which inherits the properties (methods and attributes) from one are more super classes.
- Inheritance is a powerful feature of Object Oriented Programming.
- Inheritance can facilitate code reuse, since you can customize the behavior of parent classes without having to modify them.
- Disadvantage of inheritance is that it make programs difficult to read.
  - When a method is invoked, it is sometimes not clear where to find its definition.
  - The relevant code may be scattered among several modules.
- In Inheritance base class and child classes are tightly coupled. Hence If you change the code of parent class, it will get affects to the all the child classes.



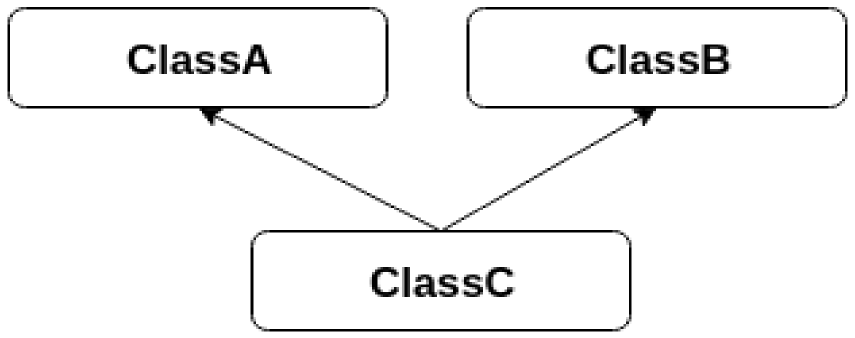
Single Inheritance



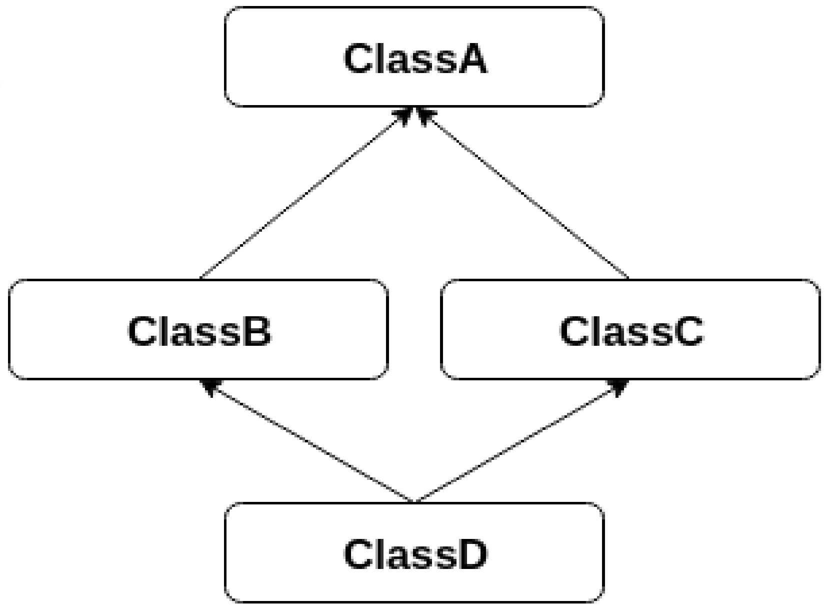
Multilevel Inheritance



Hierarchical Inheritance



Multiple Inheritance



Hybrid Inheritance

Polymorphism

Archery Range

Polymorphism

- The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.
- A language that features polymorphism allows developers to program in the general rather than program in the specific.
- In a programming language that exhibits polymorphism, objects of classes belonging to the same hierarchical tree (inherited from a common base class) may possess functions bearing the same name, but each having different behaviors
- Polymorphism is mainly divided into two types:
  - Static or Compile Time Polymorphism
    - Operator Overloading
    - Function Overloading
  - Dynamic or Runtime Polymorphism

Polymorphism

- Association of method call to the method body is known as binding.
- There are two types of binding:
  - Static Binding that happens at compile time
    - Examples:
      - Operator Overloading
      - Function Overloading
  - Dynamic Binding that happens at runtime.
    - Example:
      - Method Overriding

Method overriding

- **Method overriding** is an ability of any object-oriented programming language that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.
- When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

Function Overloading

- Two or more functions having same name but different argument(s) are known as overloaded functions
- Function Overloading refers to using the same things for different purpose
- Example:
  - int test() { }
  - int test(int a) { }
  - float test(double a) { }
  - int test(int a, double b) { }
- Python does not support Function Overloading

What are packages?

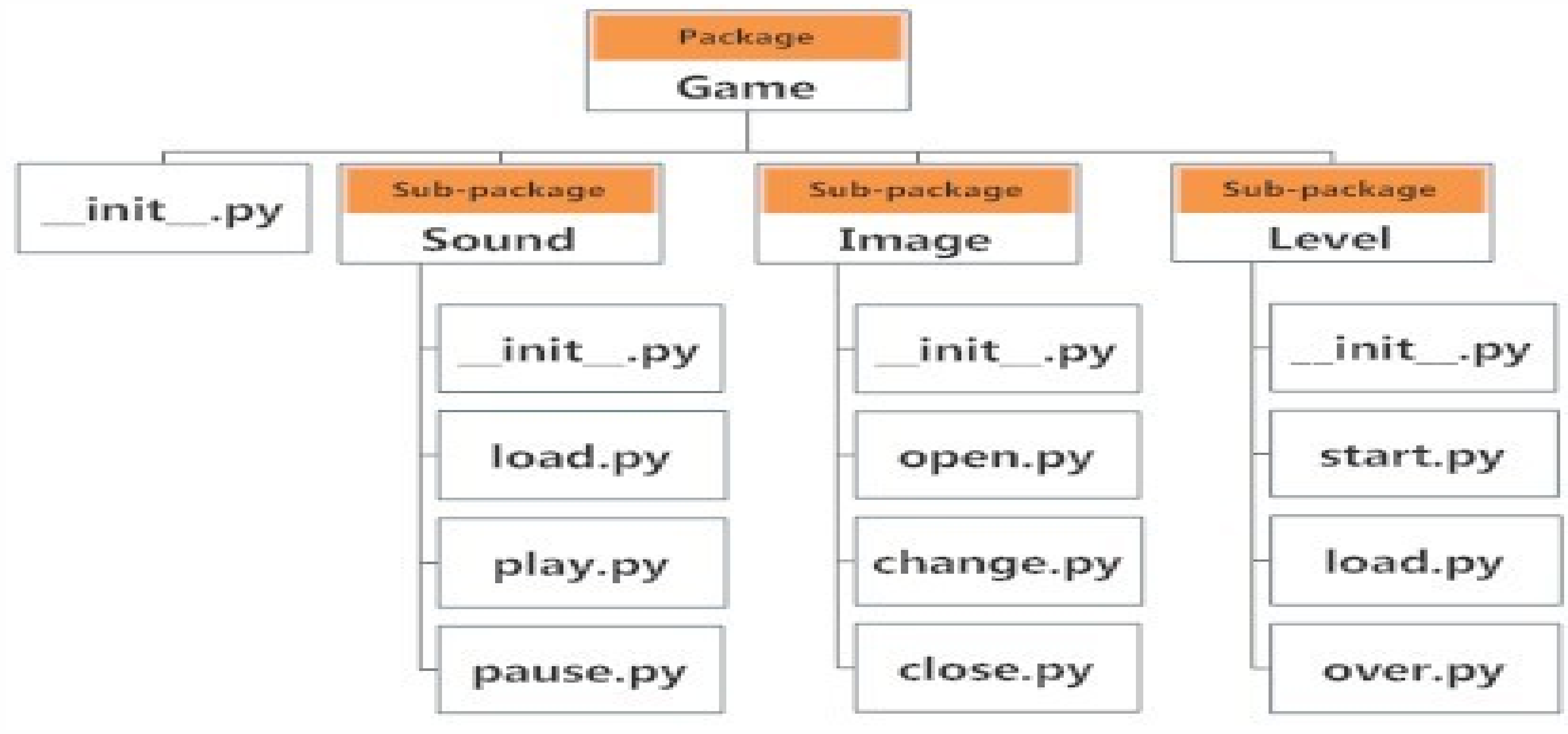
- Packages are namespaces which contain multiple packages and modules themselves. They are simply directories, but with a twist.

- Each package in Python is a directory which MUST contain a special file called **__init__.py**.

- This file can be empty, and it indicates that the directory it contains is a Python package, so it can be imported the same way a module can be imported.

- Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory. Analogous to this, Python has packages for directories and modules for files.

Importing module from a package



- import Game.Level.start

What are Numpy?

- NumPy is a Python library used for working with arrays.
- It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
- NumPy stands for Numerical Python.
- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
- Arrays are very frequently used in data science, where speed and resources are very important.
- **Example: refere class notes**

RV College of Engineering

What are Garbage Collection?

Go, change the world

- Python deletes unwanted objects (built-in types or class instances) automatically to free the memory space.
- The process by which Python periodically frees and reclaims blocks of memory that no longer are in use is called Garbage Collection.

Exception Handling

Error

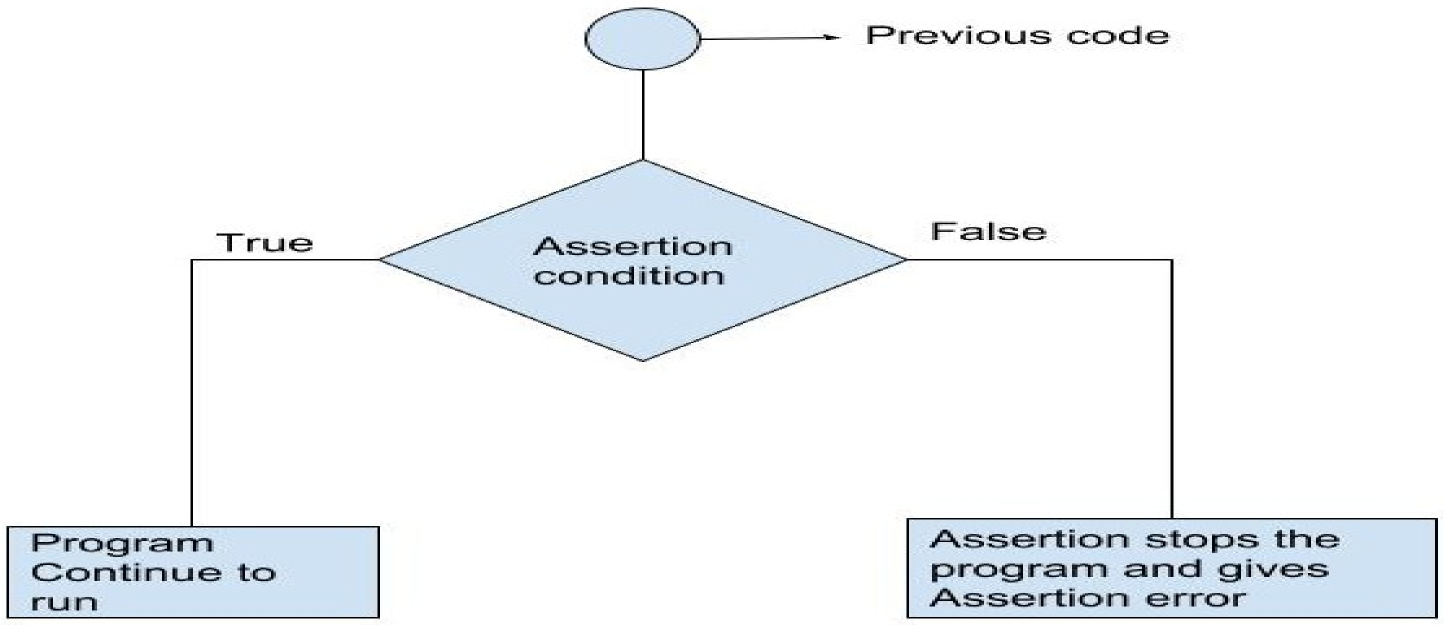
- **Errors** are the **mistakes** or **faults** in the program that causes our program to behave unexpectedly which results in abnormal working.
- There are Three types of Errors
  - **Syntax Error**
    - Syntax refers to the structure of a program and the rules about that structure
  - **Semantic Error**
    - it will run successfully,
    - The computer will not generate any error messages, but it will not do the right thing.
    - It will do something else.
  - **Runtime Error**
    - The error does not appear until you run the program.
    - These errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened.

What is Exception ?

- An Exception is an **unwanted** or **unexpected event**, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program’s instructions.
- When an exception occurs, it interrupts the flow of the program.
- If the program can handle and process the exception, it may continue running.
- If an exception is not handled, the program may be forced to quit.
- In general, when a Python script encounters a situation that it cannot cope with, it raises an exception.
- An exception is a **Python object** that represents an **error**

Assertion

- It is also a debugging tool
- it brings the program on halt as soon as any error is occurred and shows on which point of the program error has occurred.
- Assertions are simply boolean expressions that checks if the conditions return true or not. If it is true, the program does nothing and move to the next line of code. However, if it's false, the program stops and throws an error.



What are Magic Methods?

- Magic methods in Python are the special methods that start and end with the double underscores.
- They are also called dunder methods.
- Dunder or magic methods in Python are the methods having two prefix and suffix underscores in the method name.
- Dunder here means “Double Under (Underscores)”.
- These are commonly used for operator overloading.
- Magic methods are not meant to be invoked directly by you, but the invocation happens internally from the class on a certain action.
- Few examples for magic methods are: __init__, __add__, __len__, __repr__ etc.
- Example,
  - when you add two numbers using the + operator, internally, the __add__() method will be called.

What are Magic Methods?

- __str__ function is supposed to return a human-readable format, which is good for logging or to display some information about the object.
- the __repr__ function is supposed to return an “official” string representation of the object, which can be used to construct the object again. Let’s look at some examples below to understand this difference in a better way.

Modules

- Collection of related functions, variables, classes.
- Module is file of Python code
- Module is a Python object

Garea.py

```
#Module for geomitrical area calculations
def Tarea(length, base): #function to find triangle area
        area=0.5*length*base
        print(area)
def Carea(radius): # returns areaof circle
         area=3.142*pow(radius,2)
         return area
def Sarea(length):
        area=length*length
        print("Area of squarw is",area)
```

```
>>> import Garea
>>> Garea.Tarea(3,2)
3.0
>>> Garea.Sarea(3)
Area of squarw is 9
>>> print(Garea.Carea(3))
28.278
>>> print(Garea.Sarea(3))
Area of squarw is 9
None
>>> type (Garea)
<class 'module'>
```

Importing Modules

- Modules are like header files
- Module can be imported and used in script or in intractive mode

```
#Demo of importing module (modu_import.py)
import Garea
print("1:Area of circle\n","2:Area of Triangle\n","3:Area
of Square\n")
ch=int(input("Enter u r choice"))
if ch==1:
    r=float(input("Enter radius"))
    print(Garea.Carea(r))
if ch==2:
    l=float(input("Enter length"))
    b=float(input("Enter base"))
    Garea.Tarea(l,b)
if ch==3:
    s=float(input("Enter Side"))
    Garea.Sarea(s)
```

```
>>>
1:Area of circle
 2:Area of Triangle
 3:Area of Square

Enter u r choice3
Enter Side2
Area of square is 4.0
>>>
```

Import Statement

- Import statement imports names (functions) from modules.
- Different syntax of import statement.

  Syntax 1: *import module1[, module2[,... moduleN]*

  It imports only the module names into current namespace. Functions can be accessed using module name

  Syntax 2: *from modname import name1[, name2[, ... nameN]]*
  It imports specific attributes from the module into namespace
  Functions can be accessed directly

- **Syntax 3: *from modname import ****
  Imports all items from the module

- **Syntax 4: *import modname as newname***
  It imports module in alias name

```
>>> import Garea
>>> Garea.Sarea(3)
Area of square is 9
```

```
>>> from Garea import Tarea
>>> Tarea(3,5)
7.5
```

```
>>> import Garea as Shapes
>>> Shapes.Sarea(2)
Area of square is 4
```