Introduction to Python

Prof. Soumya Patil

Assistant Professor,

Dept. of CSE, DSCE, Bengaluru

Python – a mysterious name

Python is a widely used general-purpose, high level programming language. It was initially designed by Dutch programmer **Guido van Rossum in 1991.**

The name Python comes from an old BBC television comedy sketch series called Monty Python's Flying Circus. When Guido van Rossum was creating Python, he was also reading the scripts of Monty Python. He thought the name Python was appropriately short and slightly mysterious.

Python versions

Python was first released on February 20, 1991 and later on developed by Python Software Foundation.

Major Python versions are – **Python 1**, **Python 2 and Python 3**.

- On 26th January 1994, **Python 1.0** was released.
- On 16th October 2000, **Python 2.0** was released with many new features.
- On 3rd December 2008, **Python 3.0** was released with more testing and includes new features.

Latest version - On 2nd October 2023, **Python 3.12** was released.

Now **Python 3.13, Note that Python 3.9+ *cannot* be used on Windows 7 or earlier.**

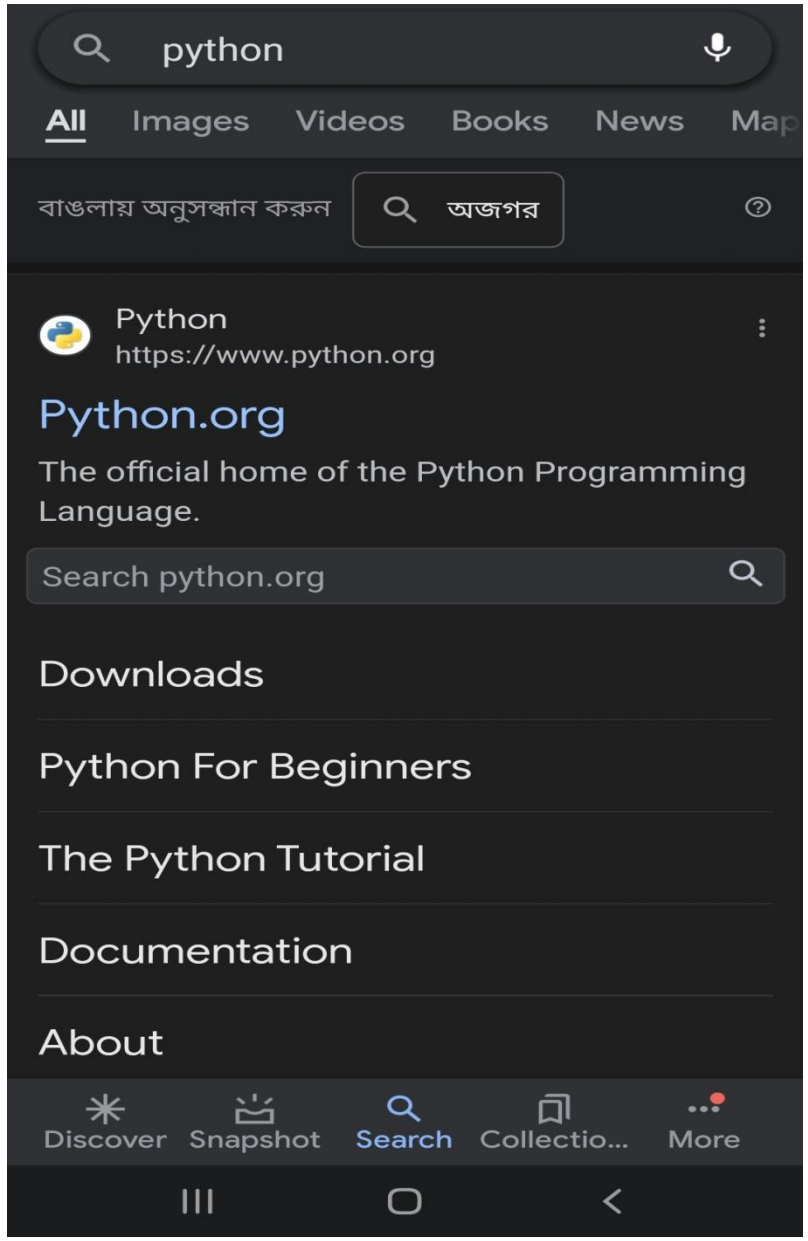
To check your Python version

i) For Linux OS type **python -V** in the terminal window.

ii) For Windows and MacOS type import sys print(sys.version) in the interactive shell.

Why should we learn Python?

- Python is a higher level programming language.
- Its syntax allows programmers to express concepts in fewer lines of code.
- Python is a programming language that lets you work quickly and integrate systems more efficiently.
- One can plot figures using Python.
- One can perform symbolic mathematics easily using Python.
- It is available freely online.



Searching for Python

Downloading Python

Python Interpreter

The program that translates Python instructions and then executes them is the Python interpreter. When we write a Python program, the program is executed by the Python interpreter. This interpreter is written in the C language.

There are certain online interpreters like

https://ide.geeksforgeeks.org/,

http://ideone.com/ ,

http://codepad.org/

that can be used to start Python without installing an interpreter.

Python IDLE

Python interpreter is embedded in a number of larger programs that make it particularly easy to develop Python programs. Such a programming environment is IDLE

(**Integrated Development and Learning Environment**).

It is available freely online. For Windows machine IDLE (Integrated Development and Learning Environment) is installed when you install Python.

Running Python

There are two modes for using the Python interpreter:

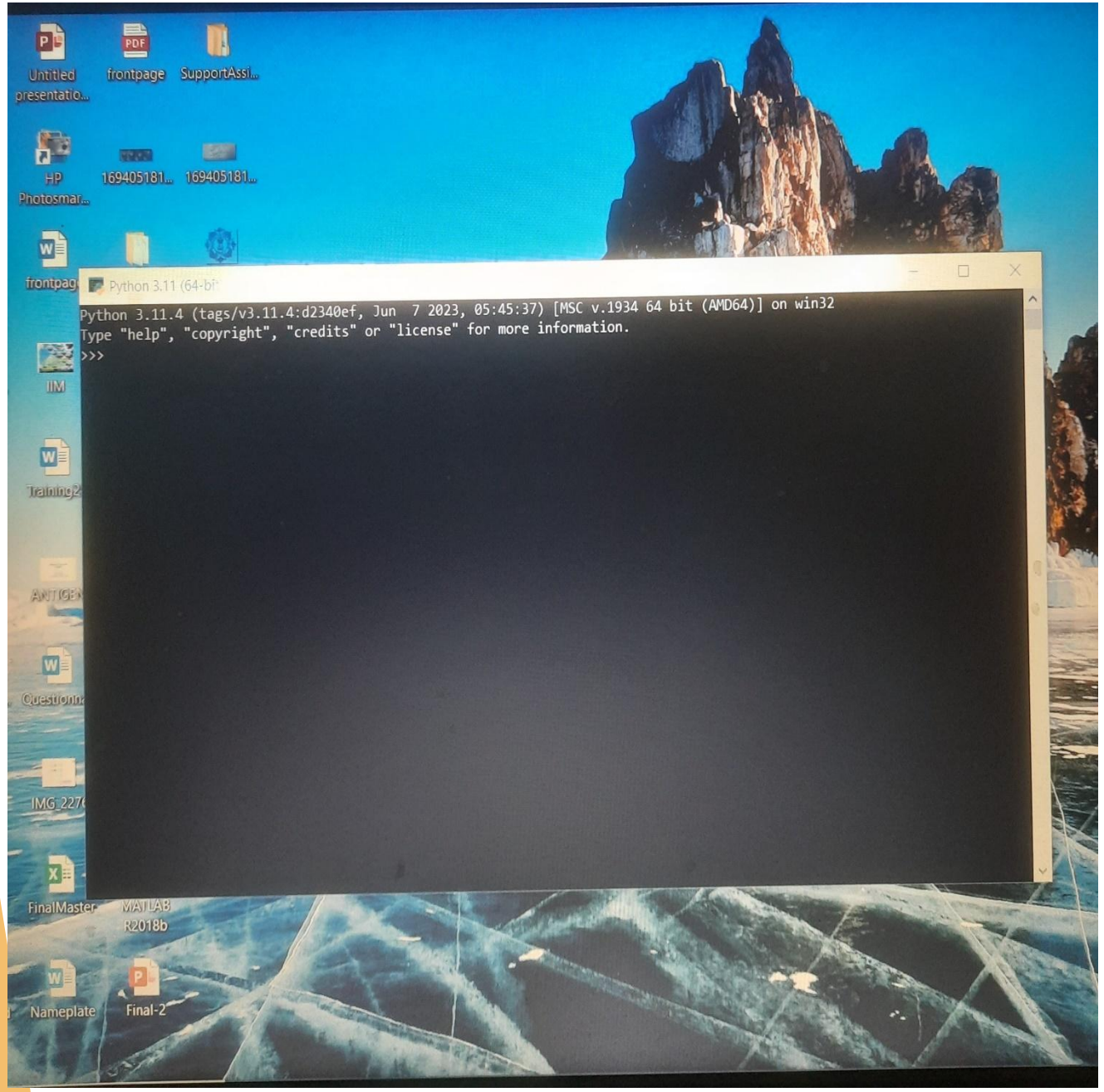
1) Interactive Mode

2) Script Mode

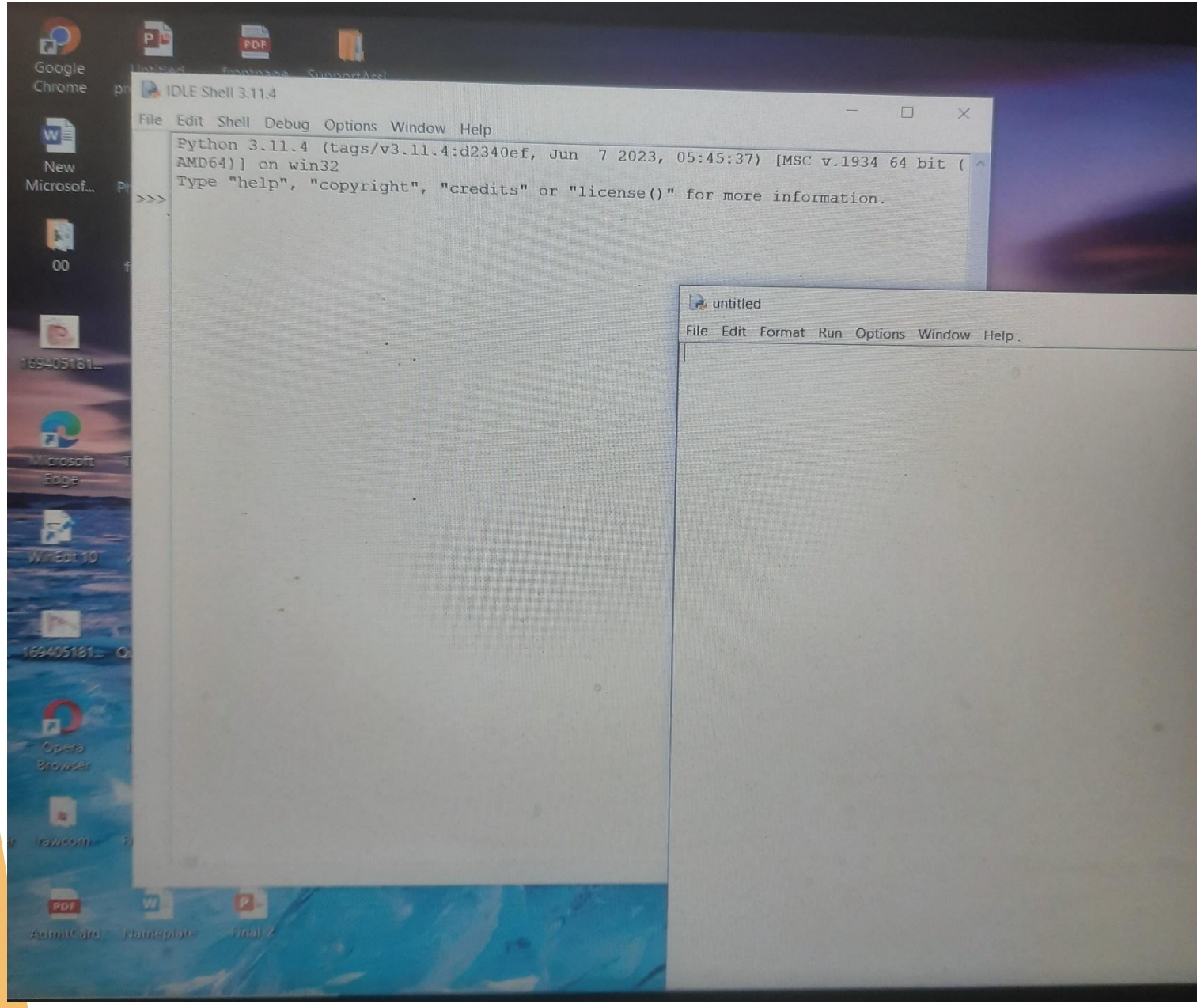
Options for running the program:

- In Windows, you can display your folder contents, and double click on madlib.py to start the program.

- In Linux or on a Mac you can open a terminal window, change into your python directory, and enter the command python madlib.py



Interactive shell


IDLE Shell 3.11.4
File Edit Shell Debug Options Window Help
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun 7 2023, 05:45:37) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
untitled
File Edit Format Run Options Window Help

IDLE shell

Running Python

1) in interactive mode:

```
>>> print("Hello Teachers")
 Hello Teachers
 >>> a=10
>>> print(a)
10
>>> x=10
>>> z=x+20
>>> z
30
```

Running Python

2) in script mode:

Programmers can store Python script source code in a file with the .py extension, and use the interpreter to execute the contents of the file.

For UNIX OS to run a script file MyFile.py you have to type:

python MyFile.py

Comments

- Start comments with #, rest of line is ignored
- Can include a “documentation string” as the first line of a new function or class you define
- Development environments, debugger, and other tools use it: it’s good style to include one

```
def fact(n):
    """fact(n) assumes n is a positive
    integer and returns facorial of n."""
    assert(n>0)
    return 1 if n==1 else n*fact(n-1)
```

Assignment

- *Binding a variable* in Python means setting a *name* to hold a *reference* to some *object*
  - *Assignment creates references, not copies*
- Names in Python do not have an intrinsic type, objects have types
  - Python determines the type of the reference automatically based on what data is assigned to it
- You create a name the first time it appears on the left side of an assignment expression:

```
x = 3
```

- A reference is deleted via garbage collection after any names bound to it have passed out of scope
- Python uses *reference semantics* (more later)

Variables

One can store integers, decimals or characters in variables.

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

```
a= 100          # An integer assignment
b = 1040.23     # A floating point
c = "John"      # A string
```

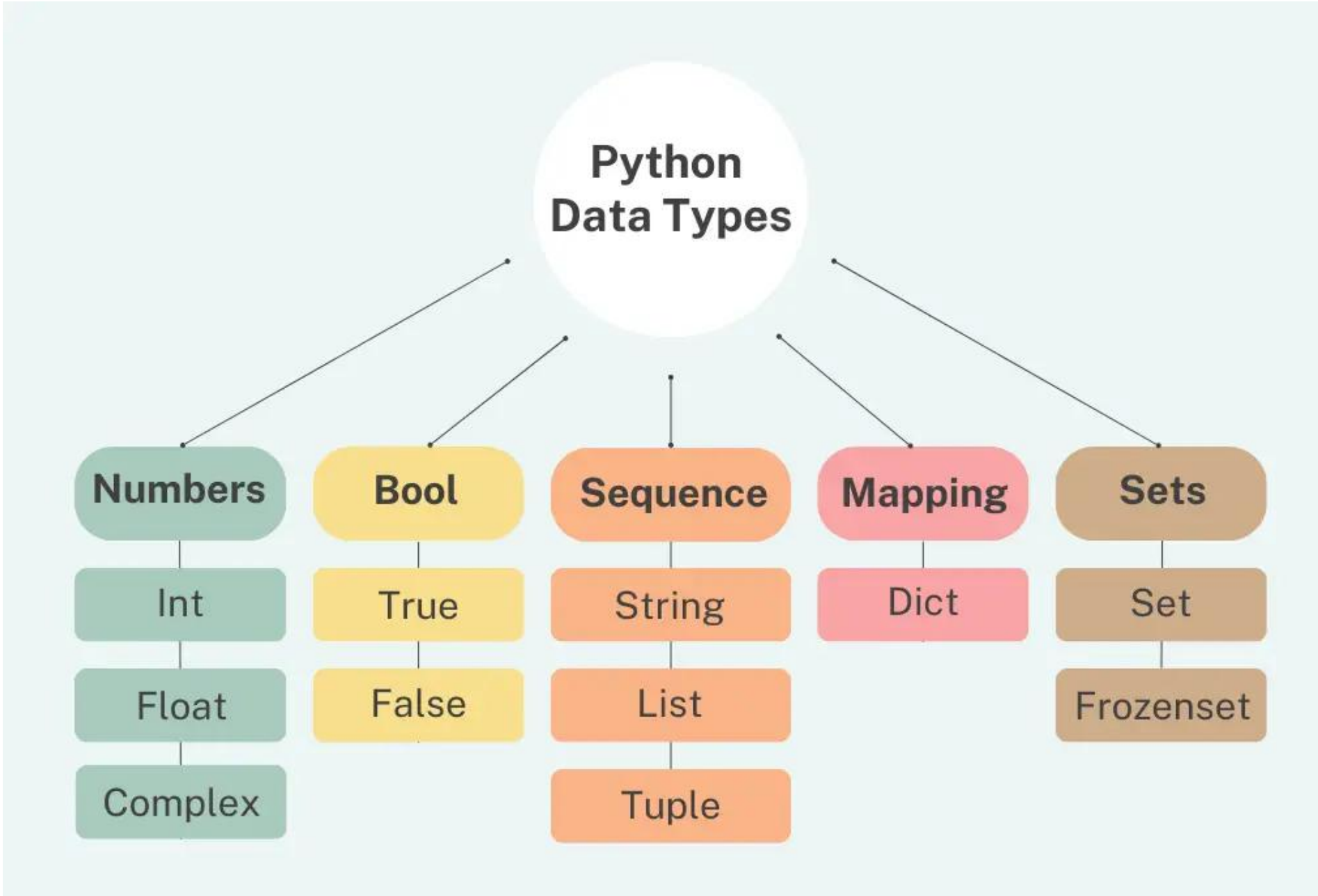
Naming Rules

- Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.

```
bob   Bob   _bob   _2_bob_   bob_2   BoB
```

- There are some reserved words:

```
and, assert, break, class, continue, def, del,
elif, else, except, exec, finally, for, from,
global, if, import, in, is, lambda, not, or, pass,
print, raise, return, try, while
```


Python
Data Types
Numbers
Int
Float
Complex
Bool
True
False
Sequence
String
List
Tuple
Mapping
Dict
Sets
Set
Frozenset

Number

- Python numbers represent data that has a numeric value. A numeric value can be an integer, a floating number or even a complex number. These values are defined as int, float and complex classes.

1. **Integers** - value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). There is no limit to how long an integer value can be.
2. **Float -** value is represented by float class. It is a real number with a floating-point representation. It is specified by a decimal point. Optionally, character e or E followed by a positive or negative integer may be appended to specify scientific notation.
3. **Complex Numbers** - It is represented by a complex class. It is specified as (real part) + (imaginary part)j. For example - 2+3j

Integer

Int:

For integer or whole number, positive or negative, without decimals of unlimited length.

```
>>> print(2465635468765)
2465635468765
 >>> print(0b10)         # 0b indicates binary number
2
>>> print(0x10)         # 0x indicates hexadecimal number
16
 >>> a=11
>>> print(type(a))
<class 'int'>
```

Float

Float:

Float, or "floating point number" is a number, positive or negative.

Float can also be scientific numbers with an "e" to indicate the power of 10.

```
 >>> y=2.8
>>> y
2.8
>>> print(0.00000045)
4.5e-07
>>> y=2.8
>>> print(type(y))
<class 'float'>
```

Complex Number

```
>>> c = 2 + 4j
>>> print(c)
(2+4j)

>>>c = 2 + 4j
>>> print(type(c))
<class 'complex'>
```

2. Sequence Data Types

The sequence Data Type is ordered collection of similar or different Python data types.

Sequences allow storing of multiple values in an organized and efficient fashion.

There are several sequence data types of Python:

1. **Strings**
2. **List**
3. **Tuple**

Strings

- Python Strings are arrays of bytes representing Unicode characters.
- In Python, there is no character data type, a character is a string of length one. It is represented by str class.
- Strings in Python can be created using single quotes, double quotes or even triple quotes.
- We can access individual characters of a String using index.

String

```
s =“Python Programming”
print(s)

print(type(s))

print(s[1])
print(s[2])
print(s[-1])
```

Lists

- Lists are just like arrays, which is an **ordered** collection of data.
- It is very flexible as items in a list do not need to be of the same type(heterogenous).
- Lists are used to store **multiple items** in a single variable.
- List items are **indexed,** the first item has index [0], the second item has index [1] etc.
- Lists are **Mutable**

List

1

Lists in Python are like arrays in C, but lists can contain data of different types.

2

The things put away in the rundown are isolated with a comma (,) and encased inside square sections [].

Lists

- **Creating a List in Python**

Lists in Python can be created by just placing sequence inside the square brackets[].

```
# Empty list
a = []

# list with int values
a = [1, 2, 3]
print(a)
```

- **Acessing a List in Python**

```
colors = ['red', 'blue', 'green']
print(colors[0])
print(colors[2])
print(len(colors))

# list with mixed int and string
list1 = ["abc", 34, True, 40.7, "male"]
```

#list of having only integers

- a= [1,2,3,4,5,6]
- print(a)

#list of having only strings

- b=[“Hello",“Ram", “Raghav"]
- print(b)

#list of having both integers and strings

- c= [“Hi",“Good",1,2,3,“Morning"]
- print(c)

#index are 0 based. this will print a single character

- print(c[1]) #this will print “Good" in list c

Lists are mutable

```
>>> x = ['abc', 23, 4.34, 23]
>>> x[1] = 45
>>> x
['abc', 45, 4.34, 23]
```

- We can change lists *in place.*
- Name *x* still points to the same memory reference when we're done.

Tuples

- Tuple is an **ordered** collection of Python objects.
- The only difference between a tuple and a list is that tuples are **immutable**.
- Tuples **cannot be modified** after it is created.

Tuple

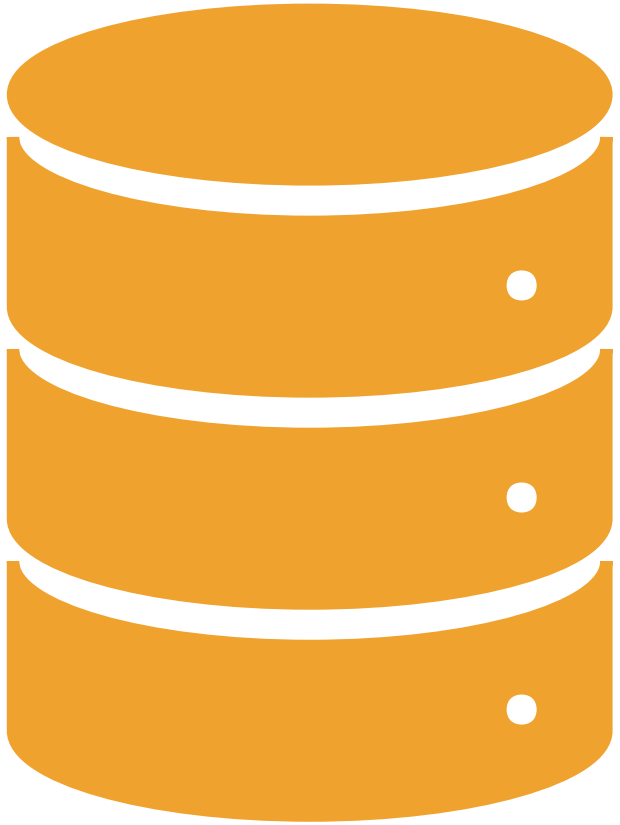
tuple having only integer type of data.

- a=(1,2,3,4)
- print(a) #prints the whole tuple

tuple having multiple type of data.

- b=("hello", 1,2,3,"go")
- print(b) #prints the whole tuple

#index of tuples are also 0 based



Tuples are immutable

```
>>> t = ('abc', 23, 4.34, 23)
>>> t[1] = 46

Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    t[1]=46
TypeError: 'tuple' object does not support item
 assignment
```

- You can't change a tuple.
- You can make a fresh tuple and assign its reference to a previously used name.

```
>>>t = ('abc', 46, 4.34, 23)
```

- *The immutability of tuples means they're faster than lists.*

LIST vs. TUPLE

IN PYTHON

Feature	List	Tuple
Definition	A collection of items that can be changed (mutable)	A collection of items that cannot be changed (immutable)
Syntax	Uses square brackets [ ]	Uses parentheses ( )
Mutability	Can change (add, remove, update) items	Cannot change items after creation
Size	Larger size because it is more flexible	Smaller size due to immutability
Performance	Slower compared to tuple	Faster due to immutability
Use Case	Use when you need to modify the data	Use when the data should stay constant
Example	my_list = [1, 2, 3]	my_tuple = (1, 2, 3)

Dictionary

- Python dictionary is a data structure that stores the value in key: value pairs.
- Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be immutable.
- It is very useful to retrieve data in an optimized way among a large amount of data.

```
#a sample dictionary variable
    a = {1:"first name",2:"last name", "age":33}

 #print value having key=1
    print(a[1])

 #print value having key=2
    print(a[2])

 #print value having key="age"
    print(a["age"])
```

```
d = { "name": "Prajjwal", 1: "Python", (1, 2): [1,2,4] }

# Access using key
print(d["name"])

# Access using get()
print(d.get("name"))
```

Set

Sets are used to store multiple items in a single variable.

Set items are unordered, unchangeable, and do not allow duplicate values.

Example:

```
set1 = {"apple", "banana", "cherry", "apple"}

print(set1)
```

Set

- set1 = {"apple", "banana", "cherry", "apple"}

  print(set1)

- print(len(set1))

Example

String, int and boolean data types:

- set1 = {"apple", "banana", "cherry"}
  set2 = {1, 5, 7, 9, 3}
  set3 = {True, False, False}

Boolean

Python Boolean Data type is one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true) and those equal to False are falsy (false). It is denoted by class bool.

Boolean:

Objects of Boolean type may have one of two values, True or False:

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

Python Booleans

Booleans represent one of two values: `True` or `False`.

Example:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

Example

Evaluate a string and a number:

```
print(bool("Hello"))
print(bool(15))
```

Evaluate two variables:

```
x = "Hello"
y = 15

print(bool(x))
print(bool(y))
```

Most Values are True

Almost any value is evaluated to True if it has some sort of content.

Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

Example
The following will return True:

```
bool("abc")
bool(123)
bool(["apple", "cherry", "banana"])
```

Some Values are False

- In fact, there are not many values that evaluate to `False`, except empty values, such as `()`, `[]`, `{}`, `""`, the number `0`, and the value `None`. And of course the value `False` evaluates to `False`.

Example

The following will return False:

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```

Input Function

- The input() function in Python is a built-in function used to obtain user input from the console.
- It pauses the program's execution and waits for the user to type something and press Enter.

Syntax:

- *input(prompt)*

- *# Taking string input*

```
name = input("Enter your name: ")
print("Hello,", name)
```

- *# Taking integer input and converting it*

```
n=int(input("enter the number: "))
print(n)
```

- *# Taking float input directly with type casting*

```
price = float(input("Enter the price: "))
print("The price is:", price)
```

Print function

>>>type(print)

Output:

builtin_function_or_method

>>>print(‘Good morning’) or print(“Good morning”)

Output:

Good morning

>>>print(“Workshop”, “on”, “Python”) or print(“Workshop on Python”)

Output:

Workshop on Python

Print function

>>>print(‘Workshop’, ‘on’, ‘Python’, sep=’\n’)

sep=‘\n’ will put each word in a new line

Output:

Workshop

on

Python

>>>print(‘Workshop’, ‘on’, ‘Python’, sep=’, ’)

sep=‘, ’ will print words separated by ,

Output:

Workshop, on, Python

Print function

%d is used as a placeholder for integer value.

%f is used as a placeholder for decimal value.

%s is used as a placeholder for string.

```
a = 2
b = ‘tiger’
print(a, ‘is an integer while’, b, ‘is a string.’)
```

Output:

2 is an integer while tiger is a string.

Alternative way:

```
print(“%d is an integer while %s is a string.”%(a, b))
```

Output:

2 is an integer while tiger is a string.

Print function

 # printing a string

name = “Rahul”

print(“Hey ” + name)

Output:

Hey Rahul

print(“Roll No: ” + str(34)) # “Roll No: ” + 34 is incorrect

Output:

Roll No: 34

 # printing a bool True / False

print(True)

Output:

True

```
a=True
print(type(a))
```

Output:

<class ‘bool’ >

```
b=10
print(type(b))
```

Output:

<class ‘int’>

Print function

```
int_list = [1, 2, 3, 4, 5]
print(int_list)     # printing a list
```

Output: [1, 2, 3, 4, 5]

```
my_tuple = (10, 20, 30)
print(my_tuple)        # printing a tuple
```

Output: (10, 20, 30)

```
my_dict = {“language”: “Python”, “field”: “data science”}
print(my_dict)        # printing a dictionary
```

Output: {“language”: “Python”, “field”: “data science”}

```
my_set = {“red”, “yellow”, “green”, “blue”}
print(my_set)        #printing a set
```

Output: {“red”, “yellow”, “green”, “blue”}

Sequence Types

- Access individual members of a tuple, list, or string using square bracket “array” notation
- *Note that all are 0 based…*

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
>>> tu[1]      # Second item in the tuple.
 'abc'

>>> li = ["abc", 34, 4.34, 23]
>>> li[1]       # Second item in the list.
 34

>>> st = "Hello World"
>>> st[1]    # Second character in string.
 'e'
```

Positive and negative indices

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Positive index: count from the left, starting with 0

```
>>> t[1]
'abc'
```

Negative index: count from right, starting with –1

```
>>> t[-3]
4.56
```

Slicing: return copy of a subset

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Return a copy of the container with a subset of the original members. Start copying at the first index, and stop copying *before* second.

```
>>> t[1:4]
('abc', 4.56, (2,3))
```

Negative indices count from end

```
>>> t[1:-1]
('abc', 4.56, (2,3))
```

Slicing: return copy of a =subset

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Omit first index to make copy starting from beginning of the container

```
>>> t[:2]
(23, 'abc')
```

Omit second index to make copy starting at first index and going to end

```
>>> t[2:]
(4.56, (2,3), 'def')
```

Copying the Whole Sequence

- [:] makes a *copy* of an entire sequence

```
>>> t[:]
(23, 'abc', 4.56, (2,3), 'def')
```

- Note the difference between these two lines for mutable sequences

```
>>> l2 = l1 # Both refer to 1 ref,
            # changing one affects both
>>> l2 = l1[:] # Independent copies, two refs
```

- To convert between tuples and lists use the list() and tuple() functions:

```
li = list(tu)                    tu = tuple(li)
```

The ‘in’ Operator

- Boolean test whether a value is inside a container:

```
>>> t = [1, 2, 4, 5]
>>> 3 in t
False
>>> 4 in t
True
>>> 4 not in t
False
```

- For strings, tests for substrings

```
>>> a = 'abcde'
>>> 'c' in a
True
>>> 'cd' in a
True
>>> 'ac' in a
False
```

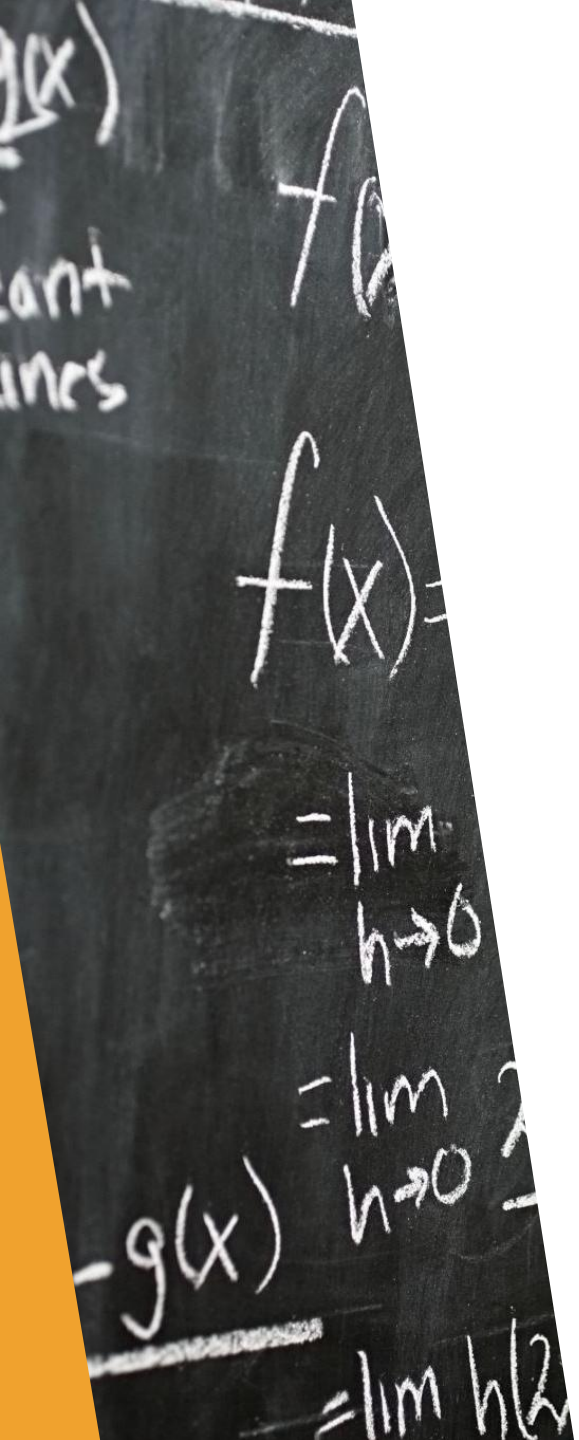
- Be careful: the *in* keyword is also used in the syntax of *for loops* and *list comprehensions*

Operators

Addition	+	Subtraction	-
Multiplication	*	Exponentiation	**
Division	/	Integer division	/ /
Remainder	%		
Binary left shift	<<	Binary right shift	>>
And	&	Or	|
Less than	<	Greater than	>
Less than or equal to	<=	Greater than or equal to	>=
Check equality	==	Check not equal	!=

Precedence of operators

Parenthesized expression	(…..)
Exponentiation	**
Positive, negative, bitwise not	+n, -n, ~n
Multiplication, float division, int division, remainder	*, /, //, %
Addition, subtraction	+, -
Bitwise left, right shifts	<<, >>
Bitwise and	&
Bitwise or	|
Membership and equality tests	in, not in, is, is not, <, <=, >, >=, !=, ==
Boolean (logical) not	not x
Boolean and	and
Boolean or	or
Conditional expression	if ….. else



Precedence of Operators

- **Examples:**
- a = 20
- b = 10
- c = 15
- d = 5
- e = 2
- f = (a + b) * c / d
- print(f)
- g = a + (b * c) / d - e
- print(g)
- h = a + b*c**e
- print(h)

Multiple Assignment

- Python allows you to assign a single value to several variables simultaneously.
- a = b = c = 1.5
- a, b, c = 1, 2, " Red“
- Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively and one string object with the value "Red" is assigned to the variable c.

Assignment

- You can assign to multiple names at the same time

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
```

This makes it easy to swap values

```
>>> x, y = y, x
```

- Assignments can be chained

```
>>> a = b = x = 2
```

Accessing Non-Existent Name

Accessing a name before it's been properly created (by placing it on the left side of an assignment), raises an error

```
>>> y

Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    y
NameError: name 'y' is not defined
>>> y = 3
>>> y
3
```

The + Operator

The + operator produces a *new* tuple, list, or string whose value is the concatenation of its arguments.

```
>>> (1, 2, 3) + (4, 5, 6)
 (1, 2, 3, 4, 5, 6)

>>> [1, 2, 3] + [4, 5, 6]
 [1, 2, 3, 4, 5, 6]

>>> “Hello” + “ ” + “World”
 ‘Hello World’
```

The * Operator

- The * operator produces a *new* tuple, list, or string that "repeats" the original content.

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

>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]

>>> "Hello" * 3
'HelloHelloHello'
```

Use of \”, \n, \t

Specifying a backslash (\) in front of the quote character in a string “escapes” it and causes Python to suppress its usual special meaning. It is then interpreted simply as a literal single quote character:

```
>>> print(" \”Beauty of  Flower\” ")
”Beauty of  Flower”
>>> print('Red  \n Blue \n Green  ')
Red
Blue
Green
>>> print("a \t  b \t  c \t  d")
a       b       c       d
```

Comments

Single-line comments begins with a hash (#) symbol and is useful in mentioning that the whole line should be considered as a comment until the end of line.

A Multi line comment is useful when we need to comment on many lines. In python, triple double quote(“ “ “) and single quote(‘ ‘ ‘)are used for multi-line commenting.
Example:

```
“““ My Program to find
    Average of three numbers ”””
a = 29      # Assigning value of a
b = 17      # Assigning value of b
c = 36      # Assigning value of c
average = ( a + b + c)/3
print(“Average value is ”, average)
```

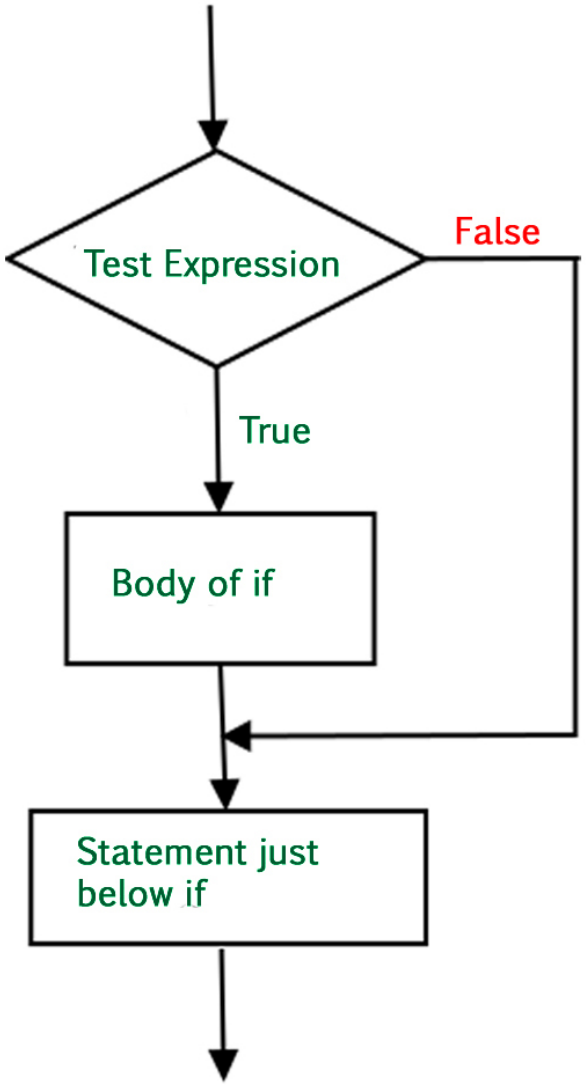
Control Flow Structures

1. Conditional if (if)
2. Alternative if (if else)
3. Chained Conditional if (if elif else)
4. While loop
5. For loop

Conditional if

#if syntax Python

if *condition*:
# Statements to execute if
# condition is true



Example:

```
a=10
if a > 9 :
    print("a is greater than 9")
```

Output:

```
a is greater than 9
```

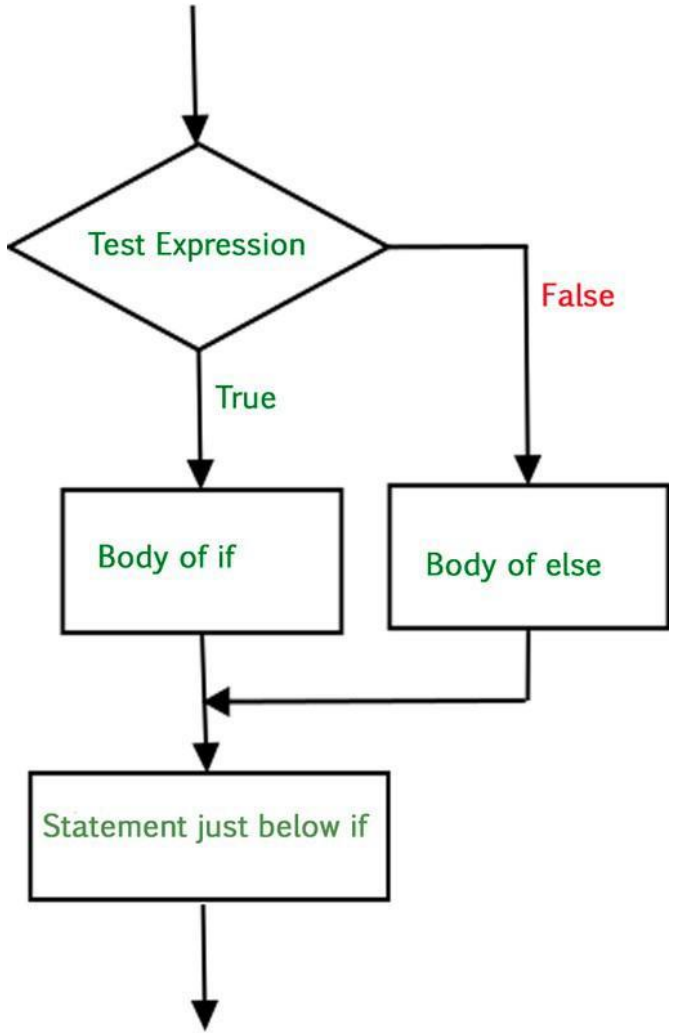
Program to check number is even or not

```
z =int(input("enter the number: "))

if z%2==0:  # True
    print(z ”, is even")

# z is even
```

else Statements

Syntax of If Else

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false
```



Write a Program to check whether a number entered by user is even or odd

```
num= int(input(“enter the number: ”))
if num%2==0 :
    print(“number is even”)
else:
    print(“number is odd”)
```

else statements(Alternative if)

Example:

```
A = int(input('Enter the marks '))
if  A >= 40:
      print("PASS")
else:
      print("FAIL")
```

Output:

Enter the marks 65

PASS

Write a program to check whether the given input number is divisible by 3 or else show a message “Number is not divisible by 3”

```
num = int(input())
if num % 3 ==0:
    print("divisible by 3")
else:
    print("Not divisible by 3")
```

 # Test if the given letter is vowel or not

```
letter = ‘o’
if letter  == ‘a’ or letter  == ‘e’ or letter  == ‘i’ or letter  == ‘o’
        or letter  == ‘u’ :
        print(letter, ‘is a vowel.’)
else:
        print(letter, ‘is not a vowel.’)
```

Output:

o is a vowel.

 # Program to find the greatest of three different numbers

```
a = int(input('Enter 1st no'))
b = int(input('Enter 2nd no'))
c= int(input('Enter 3rd no'))
if a > b:
    if a>c:
        print('The greatest no is ', a)
    else:
        print('The greatest no is ', c)
else:
    if b>c:
        print('The greatest no is ', b)
    else:
      print('The greatest no is ', c)
```

Output:

```
Enter 1st no 12
Enter 2nd no 31
Enter 3rd no 9
The greatest no is  31
```

Chained conditional if

```
# Program to guess the vegetable
 color = “green”
 if  color == “red”:
        print(‘It is a tomato.’)
elif  color ==  “purple”:
        print(‘It is a brinjal.')
elif  color == “green”:
        print(‘It is a papaya. ')
else:
        print(‘There is no such vegetable.’)
```

Output:

```
It is a papaya.
```

Write a program to accept marks from the user and display the grade.

```
marks = int(input('Enter total marks ‘))
total = 600      # Total marks
percentage=(marks/total)*100
if percentage > 90:
    print('Grade O')
elif percentage >80 and percentage<=90 :
    print('Grade A')
elif percentage >70 and percentage<=80 :
    print('Grade B')
elif percentage >60 and percentage<=70 :
    print('Grade C’)
elif percentage >40 and percentage<=60 :
    print('Grade D')
else:
    print('Fail‘)
```

Output:

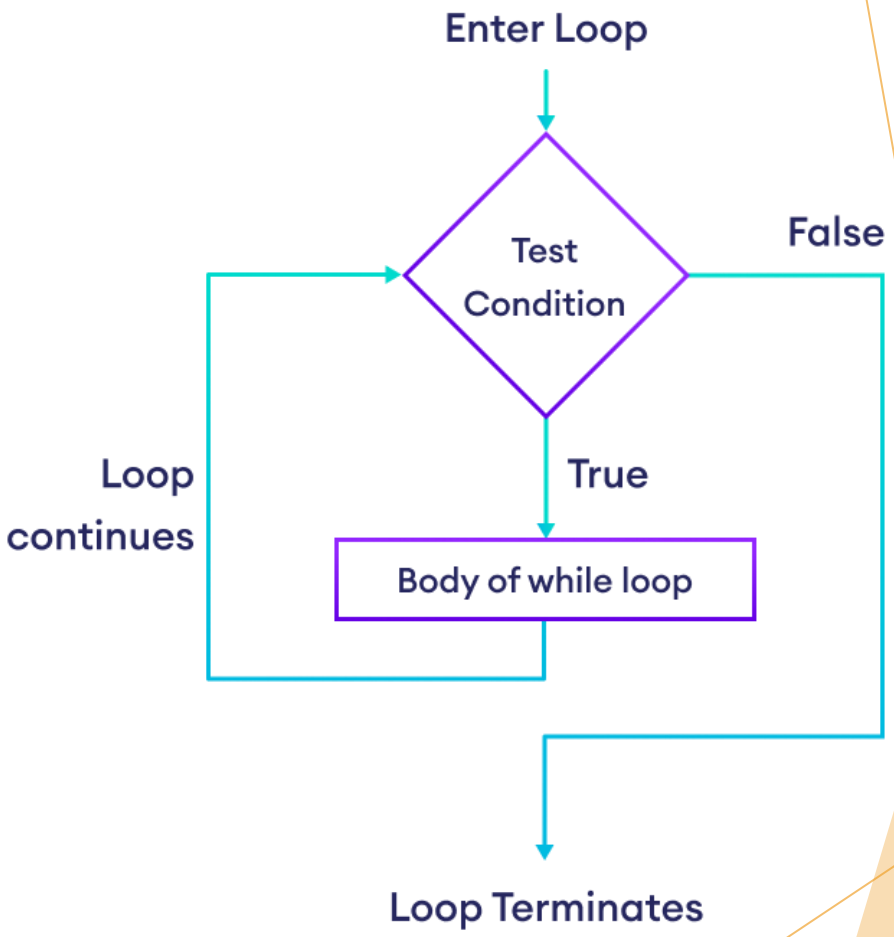
Enter total marks 312

Grade D

While loop

Syntax:

```
while expression:
    statement(s)
```



Example:

```
number = 1

while number <= 3:
    print(number)
    number = number + 1
```

Output:

```
1
2
3
```

Write a Python program to count the total number of digits in a number using a while loop

```
num = 75869
count = 0
while num != 0:
    # floor division
    # to reduce the last digit from number
    num = num // 10
    # increment counter by 1
    count = count + 1
print("Total digits are:", count)
```

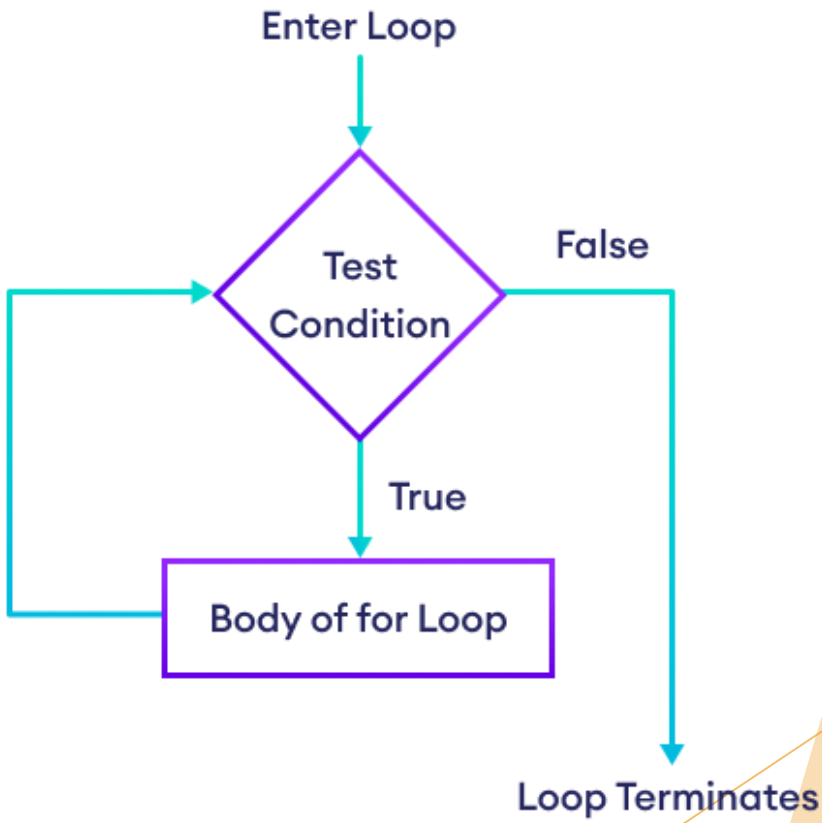
While loop

```
# Python program to find first ten Fibonacci numbers
a=1
print(a)
b=1
print(b)
i=3
while i<= 10:
    c=a+b
    print(c)
    a=b
    b=c
    i=i+1
```

For loop

Syntax:

```
for val in sequence:
        # body of the loop
```



For loop

Example:

```
# Iterating over a String
print("String Iteration")

s = “Python"
for i in s:
    print(i)
```

Output:

```
String Iteration
P
y
t
h
o
n
```

For loop

```
# Program to find the sum of a given set of numbers
numbers = [11, 17, 24, 65, 32, 69]
sum = 0
for n in numbers:
    sum = sum + n
print('The sum is ', sum)
```

Output:

```
The sum is 218
```

For loop

 # Program to find the sum of squares of first n natural numbers

```
n = int(input('Enter the last number '))
sum = 0
for i in range(1, n+1):
    sum = sum + i*i
print('The sum is ', sum)
```

Output:

```
Enter the last number 8
The sum is 204
```

 # Program to print 1, 22, 333, 444, ... in triangular form

```
n = int(input('Enter the number of rows '))
for i in range(1, n+1):
    for j in range(1, i+1):
        print(i, end='')
    print()
```

Output:

```
Enter the number of rows 5
1
22
333
4444
55555
```

 # Program to print opposite right triangle

```
n = int(input('Enter the number of rows '))
for i in range(n, 0, -1):
    for j in range(1, i+1):
        print('*', end='')
    print()
```

Output:

```
*****
****
***
**
*
```

 # Program to print opposite star pattern

```
n = int(input('Enter the number of rows '))
for i in range(0, n):
    for j in range(0, n-i):
        print(' ', end='')
    for k in range(0, i+1):
        print('*', end='')
    print('')
```

Output:

```
    *
   **
  ***
 ****
*****
```

 # Program to print A, AB, ABC, ABCD, ...

```
ch = str(input('Enter a character '))
val=ord(ch)
for i in range(65, val+1):
    for j in range(65, i+1):
        print(chr(j), end='')
    print()
```

Output:

A

AB

ABC

ABCD

ABCDE

ABCDEF

 # Program to test Palindrome numbers

```
n=int(input('Enter an integer '))
x=n
r=0
while n>0:
    d=n%10
    r=r*10+d
    n=n//10
    if x==r:
        print(x,' is Palindrome number’)
    else:
        print(x, ' is not Palindrome number')
```

 # Program to print Pascal Triangle

```
n=int(input('Enter number of rows '))
for i in range(0, n):
    for j in range(0, n-i-1):
        print(end=' ')
    for j in range(0, i+1):
        print('*', end=' ')
    print()
```

Output:

```
Enter number of rows 6
     *
    * *
   * * *
  * * * *
 * * * * *
* * * * * *
```

For Loop:

- In Python, a for loop combined with the range() function is a common way to iterate a specific number of times or through a sequence of numbers.
- The range() function generates a sequence of numbers and can be used in three ways:
- range(stop): Generates numbers from 0 up to (but not including) stop.

```
for i in range(5):
    print(i)
```

range(start, stop):
Generates numbers from start up to (but not including) stop.

- Example:

```
for i in range(2, 7):
    print(i)
```

Output:

range(start, stop, step):

Generates numbers from start up to (but not including) stop, incrementing by step in each iteration. The step can be positive for increasing sequences or negative for decreasing sequences.

- ```
  for i in range(1, 10, 2):
      print(i)
  ```

- ```
  for i in range(10, 0, -1):
      print(i)
  ```

Print characters present at an even index number

For example, str = "Python".
so your code should display
‘P’, ‘t’, ‘o’.

```
word = input('Enter word ')
print("Original String:", word)
size = len(word) # get the length of a string

# iterate each character of a string
# start: 0 to start with first character
# stop: size-1 because index starts with 0
# step: 2 to get the characters present at even index like 0, 2, 4
print("Printing only even index chars")
for i in range(0, size - 1, 2):
    print("index[", i, "]", word[i])
```

Solution :2

```
word = input('Enter word ')
print("Original String:", word)

# using list slicing
# convert string to list
# pick only even index chars
x = list(word)
for i in x[0::2]:
    print(i)
```

Break and Continue

In Python, break and continue statements can alter the flow of a normal loop.

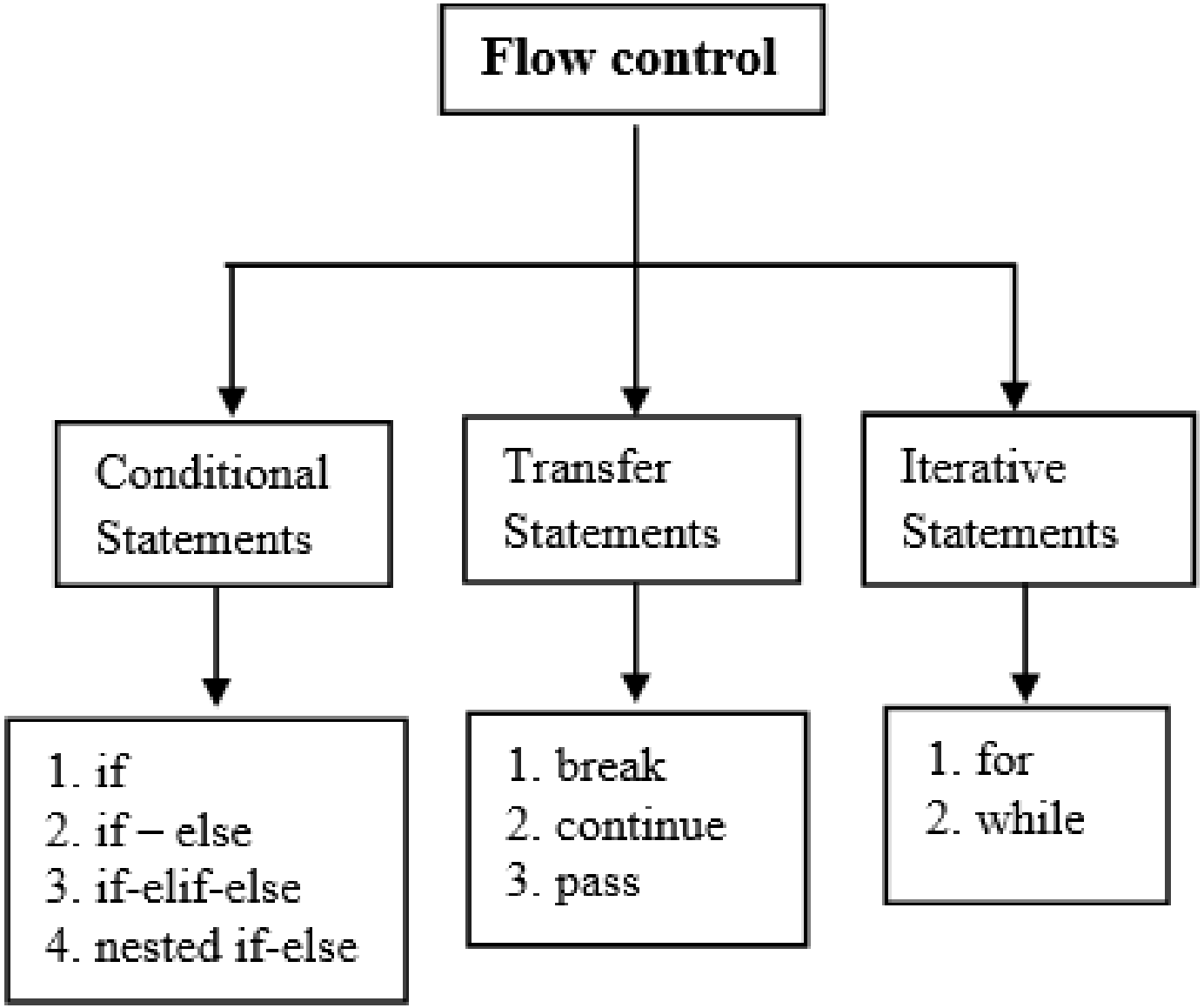
```
# Searching for a given number
numbers = [11, 9, 88, 10, 90, 3, 19]
for num in numbers:
    if(num==88):
        print("The number 88 is found")
        break
```

Output:

```
The number 88 is found
```

Break and Continue

```
# program to display only odd numbers
for num in [20, 11, 9, 66, 4, 89, 44]:
    # Skipping the iteration when number is even
    if num%2 == 0:
        continue
        # This statement will be skipped for all even numbers
    else:
        print(num)
```


Flow control
Conditional Statements
Transfer Statements
Iterative Statements
1. if
2. if – else
3. if-elif-else
4. nested if-else
1. break
2. continue
3. pass
1. for
2. while

Python Exit Commands

- Exit commands in Python refer to methods or statements used to terminate the execution of a Python program or exit the Python interpreter.
- The commonly used exit commands include `sys.exit()`, `exit()`, and `quit()`.
- These commands halt the program or interpreter, allowing the user to gracefully terminate the execution.
  1. quit() in Python
  2. exit() in Python
  3. sys.exit() using Python
  4. os._exit() in Python

- *In interactive mode (running Python in the terminal), you can typically exit by typing exit() or quit() without parentheses.*

Python Exit Commands

The exit() in Python is defined as exit commands in python if in site.py and it works only if the site module is imported so it should be used in the interpreter only.

The quit() function works as an exit command in Python if only if the site module is imported, so it should used in the interpreter.

```
for i in range(10):
    if i == 5:
        print(quit)
        quit()
    print(i)
```

```
for i in range(10):
    if i == 5:
        print(exit)
        exit()
    print(i)
```

Importing modules

Module

- Consider a module to be the same as a code library.
- A file containing a set of functions you want to include in your application.
- You can define your most used functions in a module and import it, instead of copying their definitions into different programs.
- A module can be imported by another program to make use of its functionality.
- This is how you can use the Python standard library as well.

Built-in Modules

```
import math
num = 4
print(math.sqrt(num))
```

#You need to put this command, ‘import’ keyword along

#with the name of the module you want to import

```
import math
pie = math.pi
print("The value of pi is : ", pie)
```

Writing Modules in Python

- writing a function to add/subtract two numbers in a file **calculation.py**.

```
def add(x,y):
    return (x+y)
def sub(x,y):
    return (x-y)
```

- Create another Python script in the same directory with the name module_test.py

```
import calculation
print(calculation.add(1,2))   #Calling function defined in add module.
```

Additional Ways to Import Modules in Python

- There are more ways to import modules:

1. from .. import statement
2. from .. import * statement
3. renaming the imported module

from .. import statement

Example:

#Import only Add function from module calculator and use directly

```
from calculation import add
print(add(1,2))
```

#Import multiple function from module calculator

```
from calculation import add,sub
```

from .. import * statement

- **You can import all attributes of a module using this statement**

Example:

```
from calculation import *
print(add(1,2))
print(sub(3,2))
```

renaming the imported module

- You can rename the module

1. Give a more meaningful name
2. The module name is too large to use repeatedly.

Example:

```
import calculation as cal
print(cal.add(1,2))
```

```
import math as m
 result = m.sqrt(25)
print("Square root of 25:", result
```

Practise Programs

1. Write a program which will find all such numbers which are divisible by 7 but are not a multiple of 5,

between 2000 and 3200 (both included).

The numbers obtained should be printed in a comma-separated sequence on a single line.

2. With a given integral number n, write a program to generate a dictionary that contains (i, i*i) such that is an integral number between 1 and n (both included). and then the program should print the dictionary.

Suppose the following input is supplied to the program:8

Then, the output should be:

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64}

Practise Programs

4. Write a program which accepts a sequence of comma-separated numbers from console and generate a list and a tuple which contains every number. Suppose the following input is supplied to the program: 34,67,55,33,12,98 Then, the output should be: ['34', '67', '55', '33', '12', '98'] ('34', '67', '55', '33', '12', '98')

Hints: In case of input data being supplied to the question, it should be assumed to be a console input. tuple() method can convert list to tuple

Solution:

```
values=input()
l=values.split(",")
t=tuple(l)
print(l)
print(t)
```

5. Write a program that accepts sequence of lines as input and prints the lines after making all characters in the sentence capitalized. Suppose the following input is supplied to the program: Hello world Practice makes perfect Then, the output should be: HELLO WORLD PRACTICE MAKES PERFECT

Hints: In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

```
lines = []
while True:
    s = input()
    if s:
        lines.append(s.upper())
    else:
        break;

for sentence in lines:
    print(sentence)
```

6. Write a program that accepts a sentence and calculate the number of letters and digits. Suppose the following input is supplied to the program: hello world! 123 Then, the output should be: LETTERS 10 DIGITS 3

Hints: In case of input data being supplied to the question, it should be assumed to be a console input.

Solution:

```
s = input()
d={"DIGITS":0, "LETTERS":0}
for c in s:
    if c.isdigit():
        d["DIGITS"]+=1
    elif c.isalpha():
        d["LETTERS"]+=1
    else:
        pass
print("LETTERS", d["LETTERS"])
print("DIGITS", d["DIGITS"])
```

THANK YOU