PYTHON

- This PPT is made for an individual who is willing to learn programming or want to enter in programming /coding world. This is beginner friendly ppt. and it helps an individual to learn PYTHON programming easily and effectively.
  - Note : When someone follow this ppt. he or she must do DAILY PRACTICE TO CODE .

Shubham Patidar

- This is made by Me (2021).
- When I am in my 1st Year (2nd SEM.) of B.Tech(CSE) Degree. I got oppourtunity to taught the student of Grade 11th and 12th as "A PYTHON TUTOR" at School

shubham15362@gmail.com

What is language ????

- **The principal method of human communication, consisting of words used in a structured and conventional way and conveyed by speech, writing, or gesture**

???????
???????
?

HEY !
HELLO !!

OK sir

Tell the compter to say HELLO when I say hello

Hey.... I order you. Say HELLO after my master says HELLO

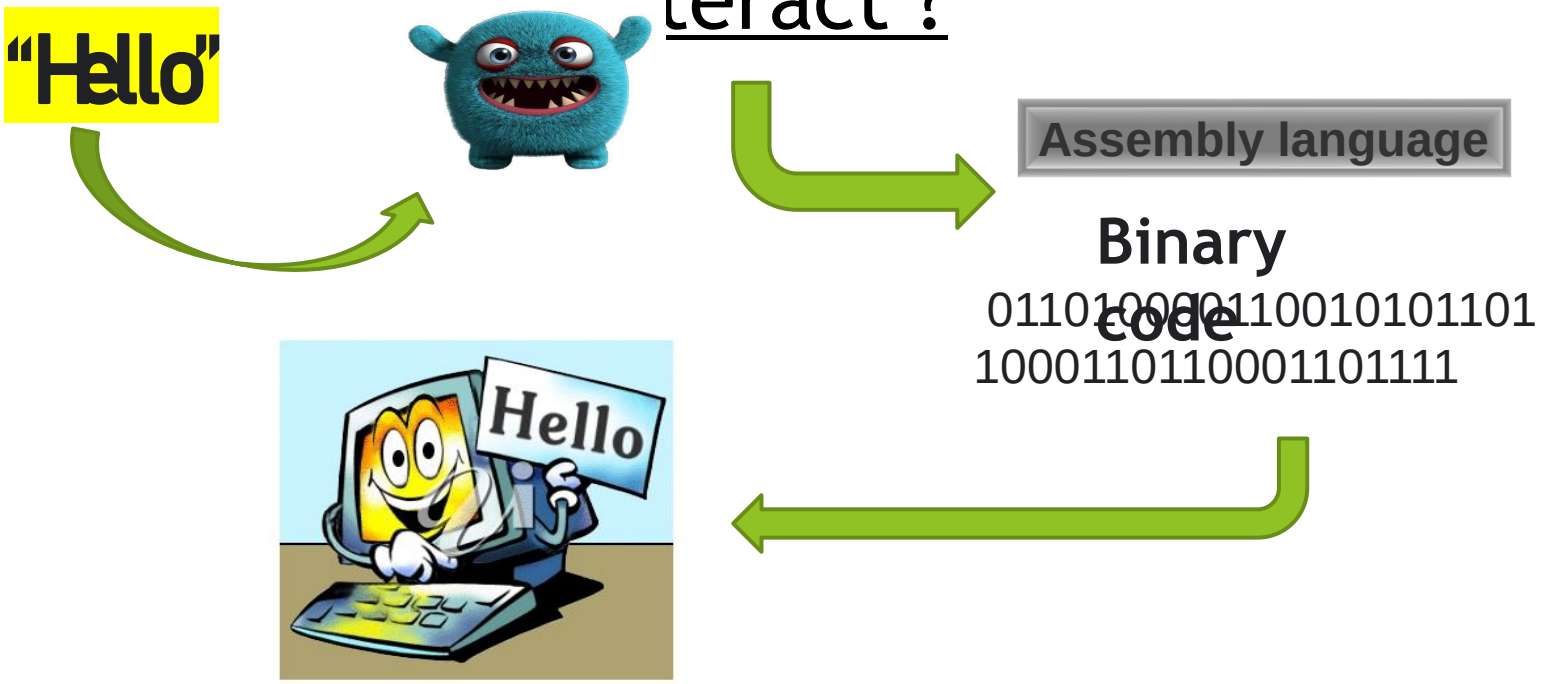
PROGRAMMING LANGUAGE

PROGRAMMER

- A **programming language** is a **computer language** that is used by **programmers** (dev elopers) to communicate with computers.

- It is a set of instructions written in any specific **language** (C, C++, Java, Python) to perform a specific task.

How computer and programmer interact ?



WHAT IS BINARY ?

- **Machine language is a low-level programming language that generally consists entirely of numbers.**
- **Binary code is basically any information represented by a sequence of 1s and 0s.**

WHY LEARN programming ?

- Attractive Salary:

- Multiple Career Opportunities:

- Develop Problem-Solving and Logical Skill:
- Develop Interpersonal Skills:
- Technologies Are Ruling the World:

- Coding is Creativity:
- Understanding of Both Sides of the Equation in Business:
- Empowering and Life-Changing Experience:

- Self interest:

python™

BY: SHUBHAM PATIDAR

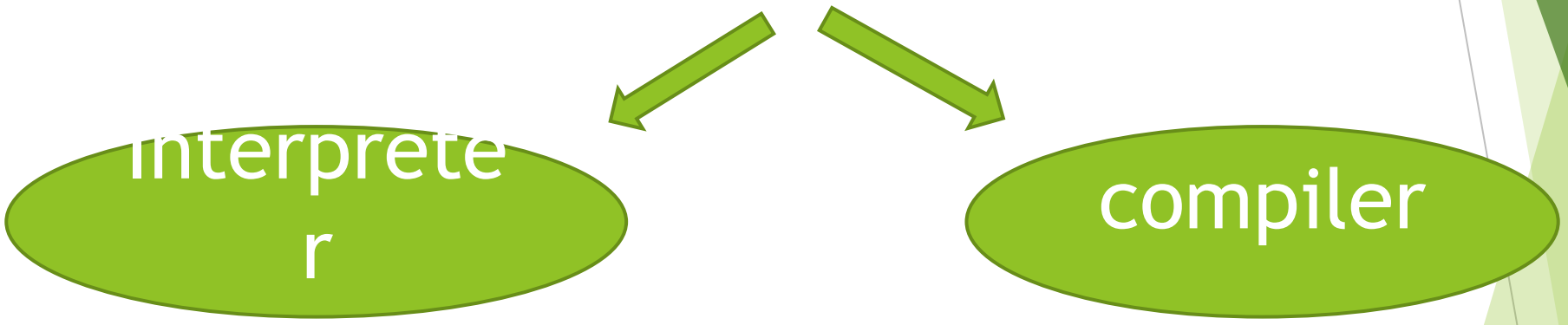
WHAT IS PYTHON ?

Python is high-level programming language which is :

- Interpreted: Python is processed at runtime by the interpreter.
- Interactive: You can use a Python prompt and interact with the interpreter directly to write your programs.
- Object-Oriented:. Python supports Object-Oriented technique of programming
- Beginner's Language: Python is a great language for the beginner-level programmers and supports the development of a wide range of applications.

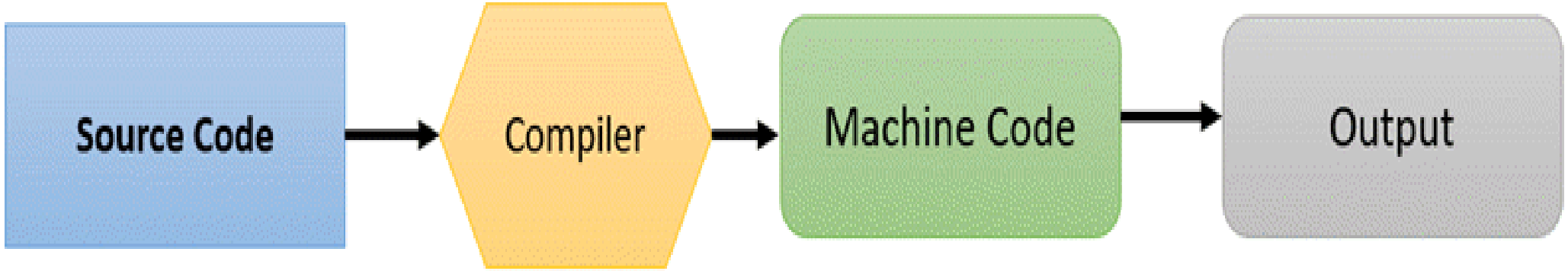
The first thing that is important to understand about Python is that it is an interpreted language.

- There are two sorts of Translator for programming languages:



Compiler	Interpreter
• A compiler takes the entire program in one go.	• An interpreter takes a single line of code at a time.
• The compiler generates an intermediate machine code.	• The interpreter never produces any intermediate machine code.
• The compiler is best suited for the production environment.	• An interpreter is best suited for a software development environment.
• The compiler is used by programming languages such as C, C ++, C #, Scala, Java, etc.	• An interpreter is used by programming languages such as Python, PHP, Perl, Ruby, etc.

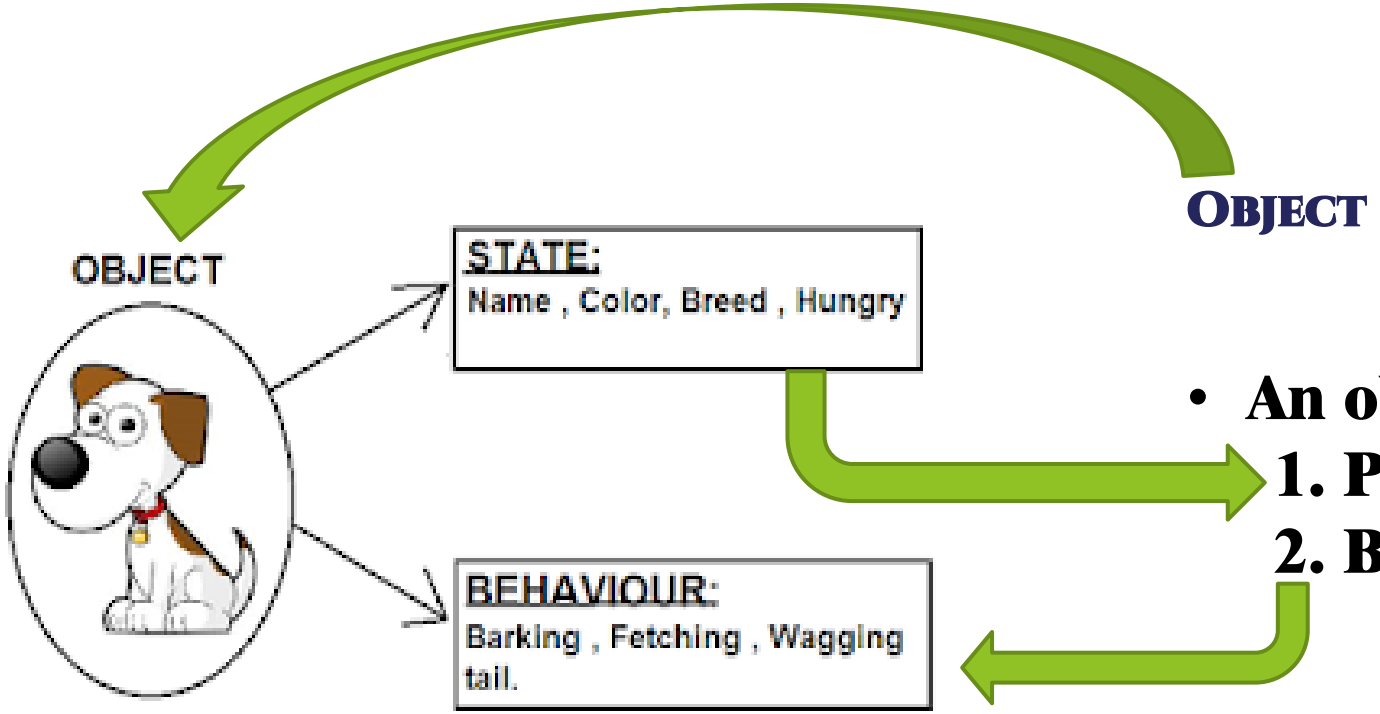
How Compiler Works



How Interpreter Works

WHAT IS Object Oriented Programming ?

- Python supports Object-Oriented technique of programming
- Means that it can model real-world entities



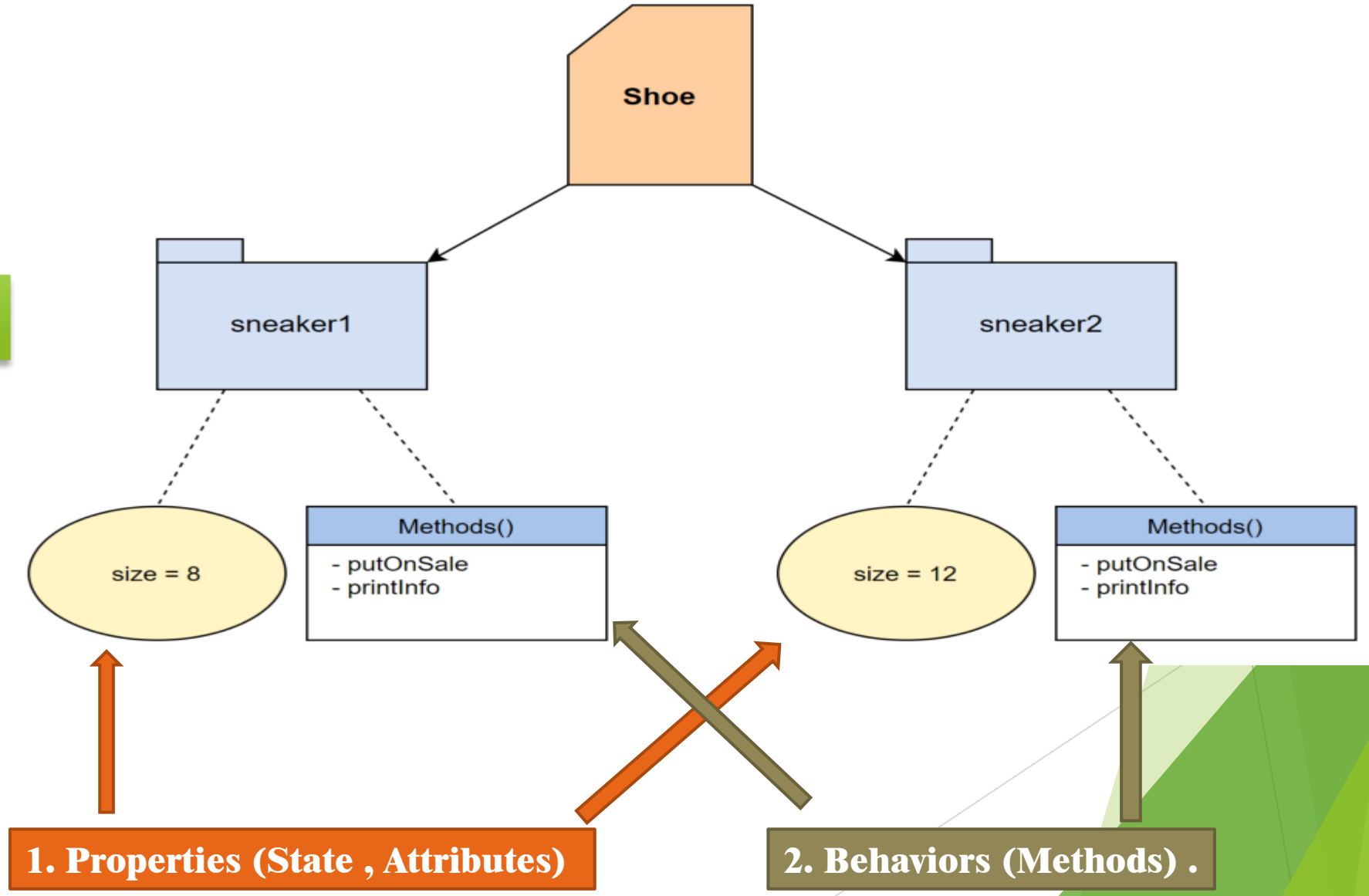
Class

OBJECT

- **An object has,**
  1. **Properties (State , Attributes)**
  2. **Behaviors (Methods) .**

CLASS

OBJECT



Python Features

- **Easy to learn:** easy to read and easy to maintain.
- **Portable**: It can run on various hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter.
- **Scalable:** Python provides a good structure and support for large programs.
- Python has support for an **interactive** mode of testing and debugging.
- Everything in Python is an **object:** variables, functions, even code. Every object has an ID, a type, and a value.

Python Standard Library is a collection of script modules accessible to a Python program to simplify the programming process and removing the need to rewrite commonly used commands.

NumPy

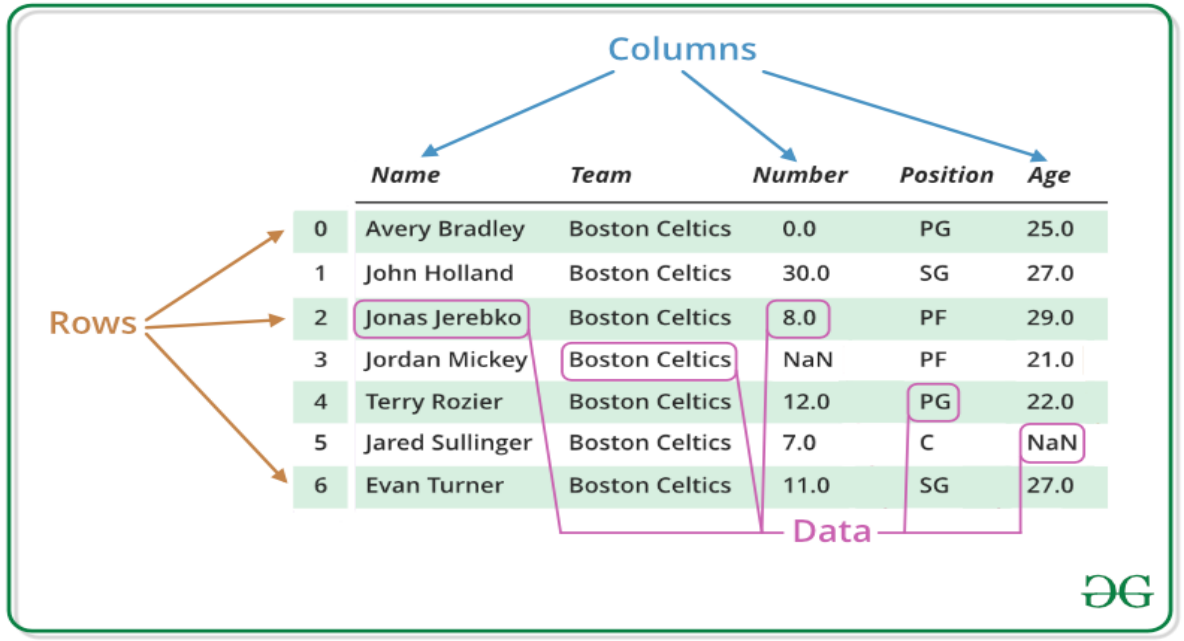
DATA MANAGEMENT

DATA MANAGEMENT

PANDAS:

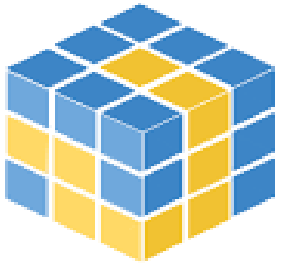
"Python Data Analysis Library "

- **Pandas is a Python library used for working with data sets.**
- **It has functions for analyzing, cleaning, exploring, and manipulating data.**



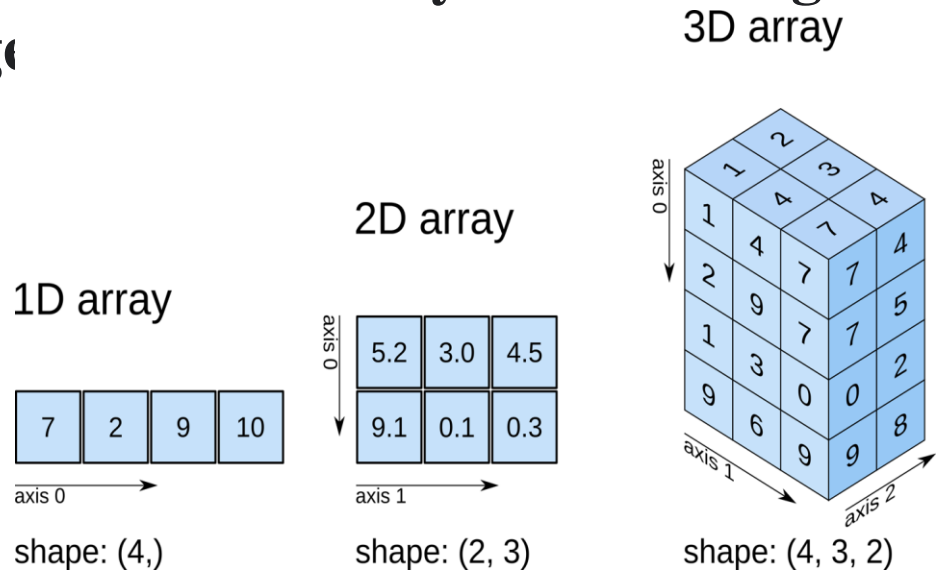
pandas

NUMPY:



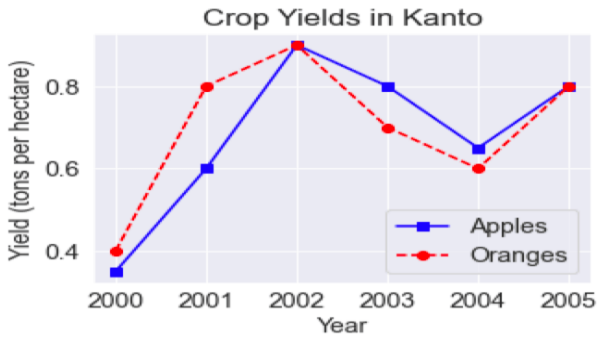
NumPy

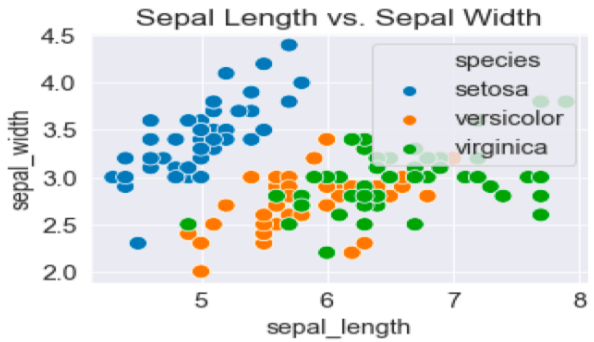
- **NumPy contains a multi-dimensional array and matrix data structures.**
- **It can be utilise to perform a number of mathematical operations on arrays such as trigonometric, statistical, and alg**

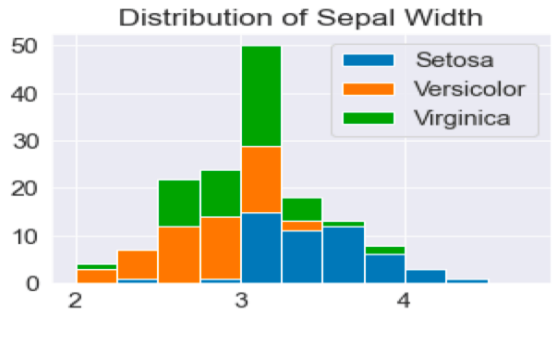


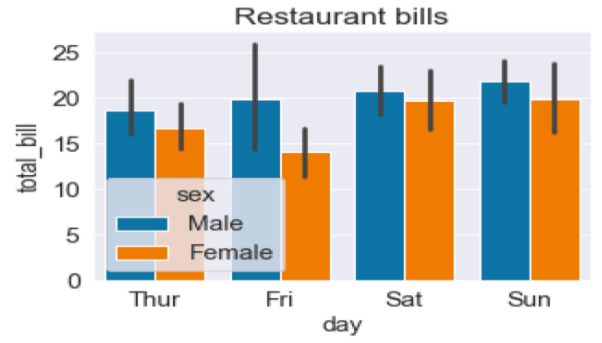
MATPLOTLIB:

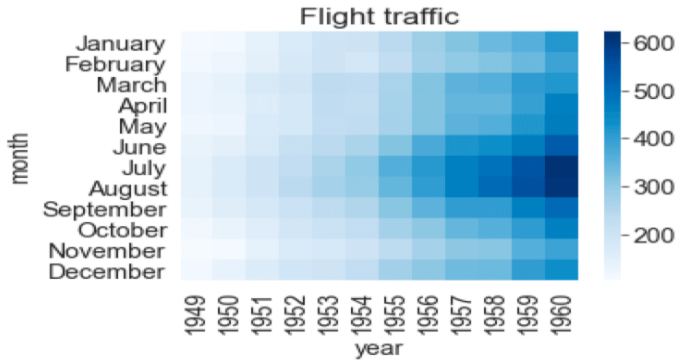
- **Matplotlib is a low level graph plotting library in python that serves as a visualization utility.**

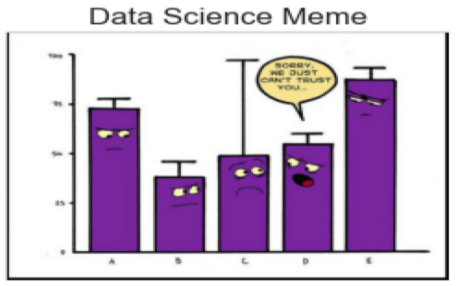












FIRST PROGRAM IN PYTHON

- The `print()` function prints the specified message to the screen,

SOURCE CODE OR INPUT

```
Print ("Hello", "how are you?")
```

OUTPUT

```
Hello how are you?
```

Java

```
public class Main
{
public static void main(String[] args) {
int a= 10, b=20;
int result = a+b;
System.out.println("The  result a+b = " + result);    }
}
```

Python

```
a=10

b=20

print ('The result a+b = ', a+b)
```

Output:

```
The result a+b = 30
```

```
The result a+b =  30
```

HISTORY OF PYTHON

- Python was conceptualized by Guido Van Rossum in the late 1980s.
- Rossum published the first version of Python code (0.9.0) in February 1991 at the CWI (Centrum Wiskunde & Informatica) in the Netherlands , Amsterdam.
- Python is derived from ABC programming language, which is a general-purpose programming language that had been developed at the CWI.
- Rossum chose the name "Python", since he was a big fan of Monty Python's Flying Circus.
- Python is now maintained by a core development team at the institute, although Rossum still holds a vital role in directing its progress.

How to work on python ?

- A program that can read Python programming statements and execute them is the Python interpreter

- Python interpreter has two modes:
  - **Interactive mode** waits for a statement from the keyboard and executes it
  - **Script mode** reads the contents of a file (Python programor Python script) and interprets each statement in the file

Interpreter Mode

- Invoke Python interpreter through Windows or command line
- >>> is the prompt that indicates the interpreter is waiting for a Python statement

```
>>> print ('Python programming is fun!')
Python programming is fun!
>>>
```

- Statements typed in interactive mode are not saved as a program

Script Mode

Writing Python Programs and Running Them in Script Mode

- Use a text editor to create a file containing the Python statements
- Save the file with a .py extension
- To run the program:

```
>>> python test.py
```

THREE KINDS OF ERRORS

- **SYNTAX ERROR :**

Some statement in the program is not a legal statement in the language.

- **RUNTIME ERROR :**

An error occurs while the program is executing, causing the program to terminate (divide by zero, etc.)

- **LOGIC ERROR :**

The program executes to completion, but gives incorrect results.

USEFUL TOOL

Python IDEs

- ✓Vim
- ✓Eclipse with PyDev
- ✓Sublime Text
- ✓Emacs
- ✓Komodo Edit
- ✓PyCharm

An integrated development environment is a software application that provides comprehensive facilities to computer programmers for software development.

An IDE normally consists of at least a source code editor, build automation tools and a debugger.

ORANISATIONS USE PYTHON

- Web Development : Google , Yahoo , Instagram
- Games : Battlefield 2 , Crystal Space
- Graphics : Walt Disney Feature Animation , Blender 3D
- Science : National Weather Service , NASA , Applied Maths
- Software Development : Nokia , Red Hat , IBM
- Education : University of California-Irvine , Sch
- Government : USA Central Intelligence Agency (CIA)

The Python Character Set

It is the set of valid characters that python understands.

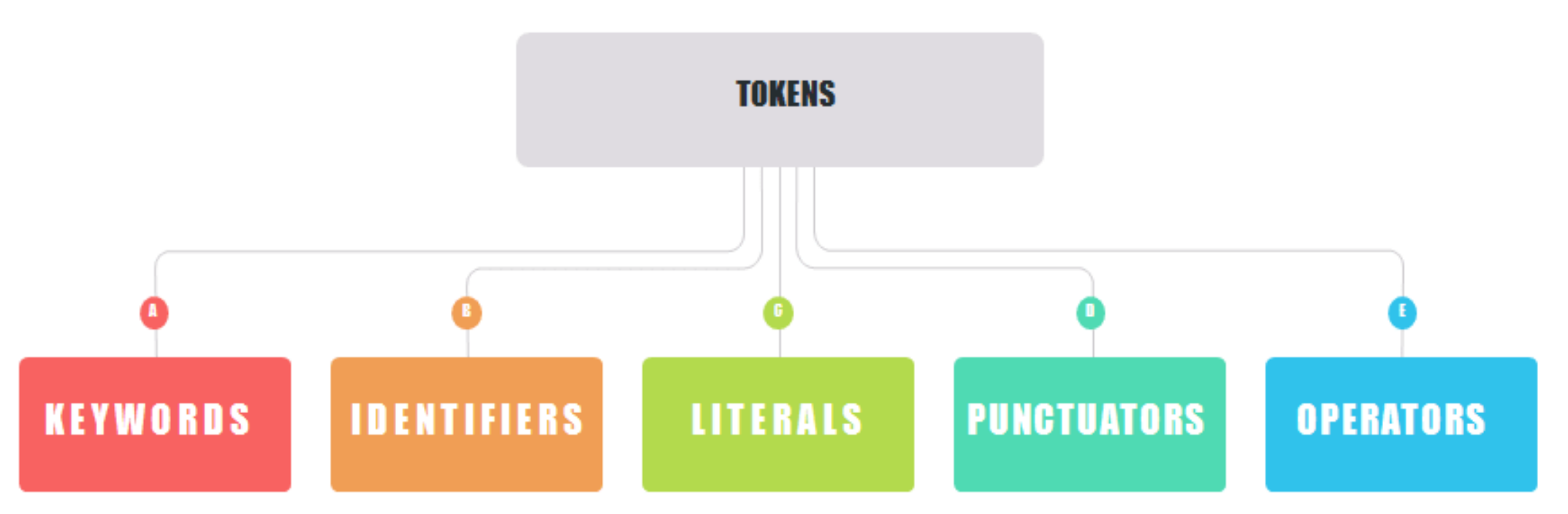
Python uses the Unicode character set.

It includes numbers from numbers like

Digits: 0-9,
Letters: a-z, A-Z
Operators: + , - , / , * , // , **
Punctuators like : (colon) , (), {}, []
Whitespaces like space, tabs, etc

WHAT ARE TOKENS ?

- Tokens are **building blocks** of a language.
- They are the smallest individual unit of a program.
- Also called lexical unit.



Keyword

- **Keywords are special words which are reserved and have a specific meaning in programming language.**
- **All keywords in Python are case sensitive. So, you must be careful while using them in your code.**

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

identifiers

- **An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.**
- **User-defined names.**

Rules for writing **IDENTIFIERS**

- Identifiers can be a combination of letters in lowercase **(a to z)** or uppercase **(A to Z)** or digits **(0 to 9)** or an underscore _.
  Names like myClass, var_1 and print_this_to_screen, all are valid example.

- An identifier cannot start with a digit. 1variable is invalid, but variable1 is a valid name.

- Keywords cannot be used as identifiers.
- We cannot use special symbols like **!**, **@**, **#**, **$**, **%** etc. in our identifier.
- An identifier can be of any length.

EXAMPLE OF IDENTIFIERS

Python Valid Identifiers Example

Let's look at some examples of valid python identifiers.

- **ab10c**: contains only letters and numbers
- **abc_DE**: contains all the valid characters
- **_:** surprisingly but Yes, underscore is a valid identifier
- **_abc**: identifier can start with an underscore

Python Invalid Identifiers Example

Let's look at some examples of invalid python identifiers.

- **99**: identifier can't be only digits
- **9abc**: identifier can't start with number
- **x+y**: the only special character allowed is an underscore
- **for**: it's a reserved keyword

LITERALS

- Often referred as Constant- Values
- Literals are data items that have a fixed value
- Python allows several kinds of literals:
  - String literals
  - Numeric literals
  - Boolean literals
  - Special literal NONE
  - Literal Collections

String literals

String literals can be forms by enclosing text in both forms quotes i.e single quotes or double quotes

Types of Strings:

a) Single-line String- Strings that are terminated within a single-line are known as Single line Strings.
Example: text1='hello'

b) Multi-line String - A piece of text that is written in multiple lines is known as multiple lines string.
There are two ways to create multiline strings:

1) Adding black slash at the end of each line.
Example:

```
text1='hello\
user'
print(text1)
```

Output:

```
'hellouser'
```

2) Using triple quotation marks:-
Example:

```
str2='''welcome
to
SSSIT'''
print (str2)
```

Output:

```
welcome
to
SSSIT
```

Numeric literal

Numeric literals are those literals which can contain digits only .
Types of numeric literal are:

- **Integer Literals**
- **Floating Point Literals**
- **Complex Literals**

•Fractional Form
•Exponent Form

The real number in exponent form has two parts:
–**Mantissa** :- The digits before the symbol E form the Mantissa part.
–**Exponent** :- The digits after the symbol E form the Exponent part. The exponent Part can only contain integers

1.33 x 105 can be represented as 1.33E5

Complex literals are of the form A+Bj.
Here A and B are real values. A is termed as real part and B is termed as imaginary part.
j represents square root of -1. Which is an imaginary number.
Example:
2.5+3.1j is a complex literal

Boolean literal

A Boolean literal can have any of the two values: True or False.

Example - Boolean Literals

```
x = (1 == True)
y = (2 == False)
z = (3 == True)
a = True + 10
b = False + 10
print("x is", x)
print("y is", y)
print("z is", z)
print("a:", a)
print("b:", b)
```

Output:

```
x is True
y is False
z is False
a: 11
b: 10
```

Special literal

Python contains one special literal i.e., **None.**

None is used to specify to that field that is not created. It is also used for the end of lists in Python.

Example - Special Literals

```
val1=10
val2=None
print(val1)
print(val2)
```

Output: 10
None

Literal collection

- ***Python provides the four types of literal collection such as***
- ***List literals.***
- ***Tuple literals.***
- ***Dict literals.***
- ***Set literals.***

List literal

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- A list can contain different data types
- Lists are used to store multiple items in a single variable
- Lists are created using square brackets:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Tuple literal

- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.

A tuple can contain different data types

Tuples are written with round brackets.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Dictionary literal

Dictionaries are used to store data values in key:value pairs.

Dictionaries cannot have two items with the same key

can be referred to by using the key name.

•Dictionary is a collection which is ordered* and changeable. No duplicate members.

```
thisdict
= {"brand": "Ford","model": "Mustang","year": 1964}
print(thisdict)
```

Set literals

- **Set** is a collection which is unordered and unindexed. No duplicate members.

Set items can be of any data type

Sets are written with curly brackets.

```
thisset={"apple", "banana", "cherry", "apple"}
print(thisset)
```

OPERATORS

Operators in general are used to perform operations on values and variables in Python.

Python language supports the following types of operators.

- **Arithmetic Operators**
- **Comparison(i.e., Relational) Operators**
- **Assignment Operators**
- **Logical Operators**
- **Bitwise Operators**
- **Membership Operators**
- **Identity Operator**

Arithmetic operator

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Comparison operator

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Assignment operator

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Logical operator

Logical operators are used to combine conditional statements:

Operator	Description	Example	
and	Returns True if both statements are true	x < 5 and x < 10	
or	Returns True if one of the statements is true	x < 5 or x < 4	
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)	

Binary 2 is 1 0
3 is 1 1

Bitwise operator

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Membership operator

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Identity operator

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example	
is	Returns True if both variables are the same object	x is y	
is not	Returns True if both variables are not the same object	x is not y	

PUNCTUATORs

Theses are symbols that are used in programming languages to organize programming- sentence structures and indicate the rhythm and emphasis of statements, expressions and program structure.

Example : ‘ “ # \ ( ) { } [ ] @ : , =

BAREBONES OF A PYTHON PROGram

EXPRESSIONS: An Expression is any legal combination of symbols that represents a value

EXAMPLE: a+3 , b<5 ,15

STATEMENT : An Statement is a programming instruction that does something i.e , some action takes place

COMMENTS: Comments are the additional readable information to clarify the source code comments in python begin with #

FUNCTIONS: A Function is a code that has a name and it can be reused (executed again) by specifying its name in program ,where needed

BLOCKS AND INDENTATION : a group of individual statements are called as Block or code-block or suite

DYNAMIC TYPing

- A variable pointing of a certain type , can be made to point to a value of different type . This is called DYNAMIC TYPING

```
X = 25
print(x)
X = “python”
print(x)
```

STATEMENTS IN PYTHON

Instructions that a Python interpreter can execute are called statements. For example, a = 1 is an assignment statement. if statement, for statement, while statement, etc. are other kinds of statements.

1. Empty Statement
2. Simple Statement (Single Statement)
3. Compound Statement

Empty Statement

A statement that does nothing. In Python an empty statement is pass statement.
It takes the following form:

Pass

Whenever Python encounters a pass statement, Python does nothing and moves to next statement in the flow of control.

Simple Statement

Any executable statement is a simple statement in Python.
For example: Name=input(“enter your name”) Compound Statement A compound statement represents a group of statements executed as a unit.

compound statement

In Python are written in a specific pattern .

It has :

-> A header line which begins with a keyword and ends with a colon.
-> A body consisting of one or more Python statements, each indented inside the header line.

Program Logic Development Tools:

Before developing the solution of a problem in terms of a program, you should read and analyze the given problem and decide about basic sub tasks needed to solve a problem and the order of these subtasks.

The various logic development tools are:

- Flowcharts
- Pseudocode
- Decision Tree

Algorithm :

An Algorithm is a step by step procedure to solve a given problem.

For example,

An algorithm for determining average of two numbers:

Step-1: Start

Step-2: Input two numbers in variable num1 & num2

Step-3: Average= (num1+num2)/2

Step-4: Print the value of Average

Step-5: Stop

It is a set of ordered and finite steps to solve a given problem

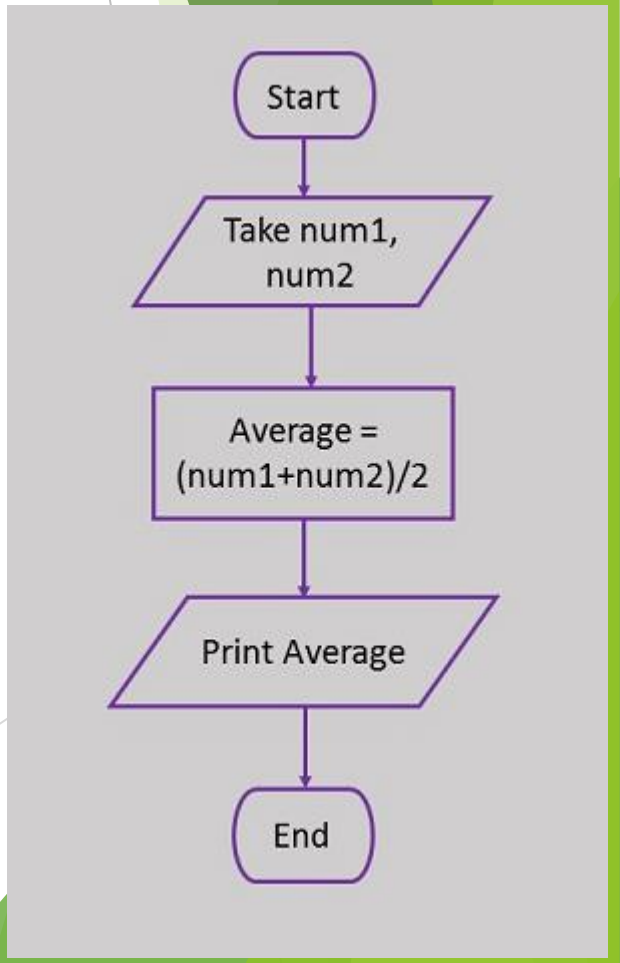
Flowchart:

- A flowchart is a graphical representation of an algorithm.
- A flowchart shows different subtasks with different symbols.

Some commonly used flowchart symbols are :

Symbol	Symbol Name	Purpose
	Start/Stop	Used at the beginning and end of the algorithm to show start and end of the program.
	Process	Indicates processes like mathematical operations.
	Input/ Output	Used for denoting program inputs and outputs.
	Decision	Stands for decision statements in a program, where answer is usually Yes or No.
	Arrow	Shows relationships between different shapes.
	On-page Connector	Connects two or more parts of a flowchart, which are on the same page.
	Off-page Connector	Connects two parts of a flowchart which are spread over different pages.

example:



Pseudocode:

Pseudo code is an informal way of programming description that does not require any strict programming language syntax or underlying technology considerations.

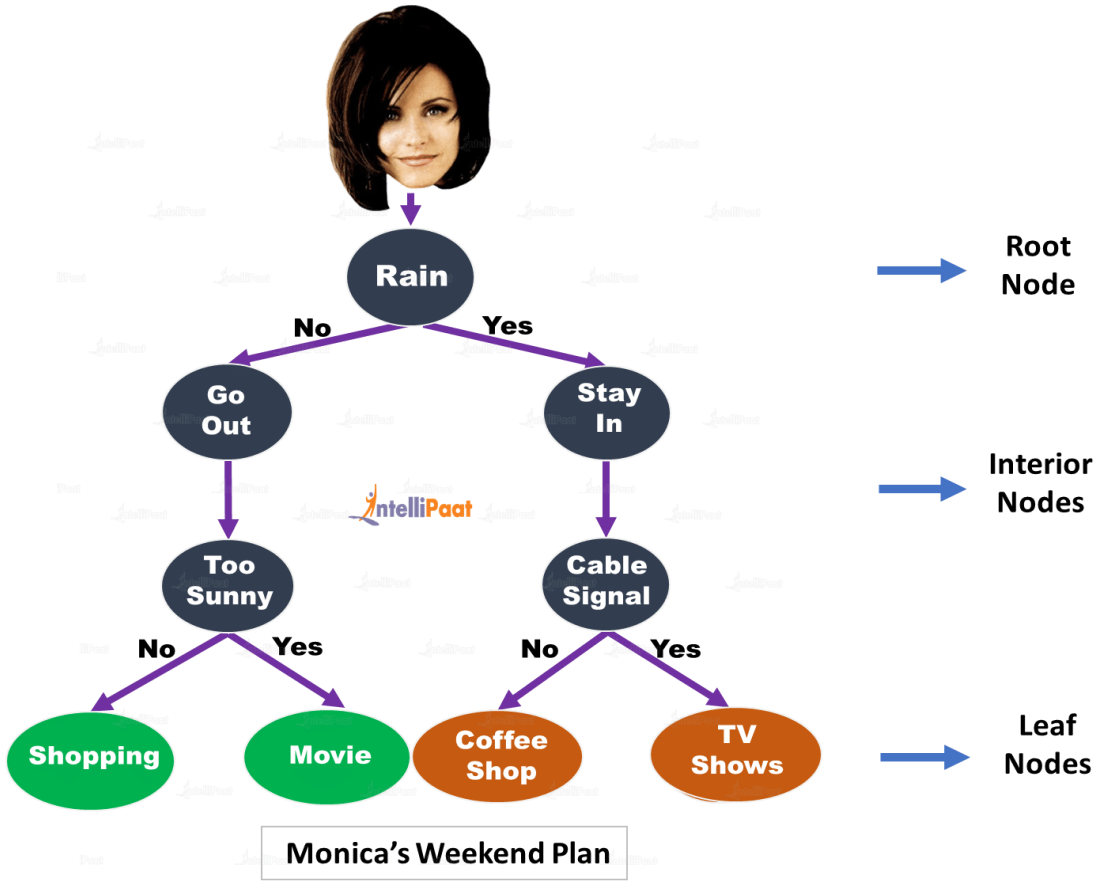
Pseudo Code for determining average of two numbers:

- **Start**
- **Input two numbers in variable num1 & num2**
- **Average= (num1+num2)/2**
- **Print the value of Average**
- **Stop**

- **It is used for creating an outline or a rough draft of a program.**
- **Pseudo code is not an actual programming language. So it cannot be compiled into an executable program.**
- **It uses short terms or simple English language syntaxes to write code for programs before it is actually converted into a specific programming language.**
- **The purpose of using pseudo code is an efficient key principle of an algorithm.**
- **It is used in planning an algorithm with sketching out the structure of the program before the actual coding takes place.**

Decision Tree:

An decision tree is a tool to represent a hierarchical structure of outcomes based on certain rules



- **Root Node:** This node gets divided into different homogeneous nodes. It represents entire sample.
- **Splitting:** It is the process of splitting or dividing a node into two or more sub-nodes.
- **Interior Nodes**: They represent different tests on an attribute.
- **Branches:** They hold the outcomes of those tests.
- **Leaf Nodes:** When the nodes can't be split further, they are called leaf nodes.
- **Parent and Child Nodes:** The node from where sub-nodes are created is called a parent node. And, the sub-nodes are called the child nodes.

Statement Flow Control :

A program's control flow is the order in which the program's code executes.

Every programming language provides constructs to support

Sequential - default mode

Selection - used for decisions and branching

Repetition - used for looping, i.e., repeating a piece of code multiple times.

1. Sequential

- Sequential statements are a set of statements whose execution process happens in a sequence.
- The problem with sequential statements is that if the logic has broken in any one of the lines, then the complete source code execution will break.

```
a=20
b=10
c=a-b
print("Subtraction is : ",c)
```

2. Selection/Decision control statements

- In Python, the selection statements are also known as *Decision control statements* or *branching statements*.
- The selection statement allows a program to test several conditions and execute instructions based on which condition is true.

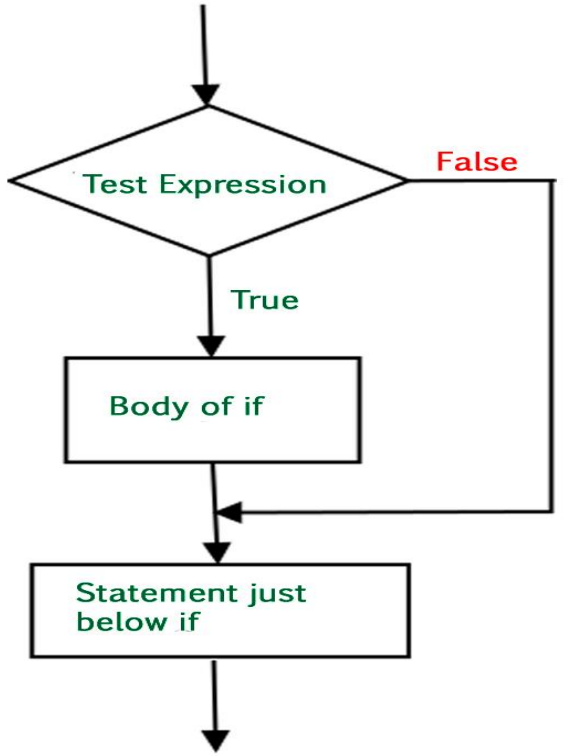
Some Decision Control Statements are:

- if
- if-else
- if-elif-else
- nested if

if

Syntax :

```
if  (condition) :
    statement
```



Example 1:

```
if grade=='A':
    print("well done")
```

Example 2:

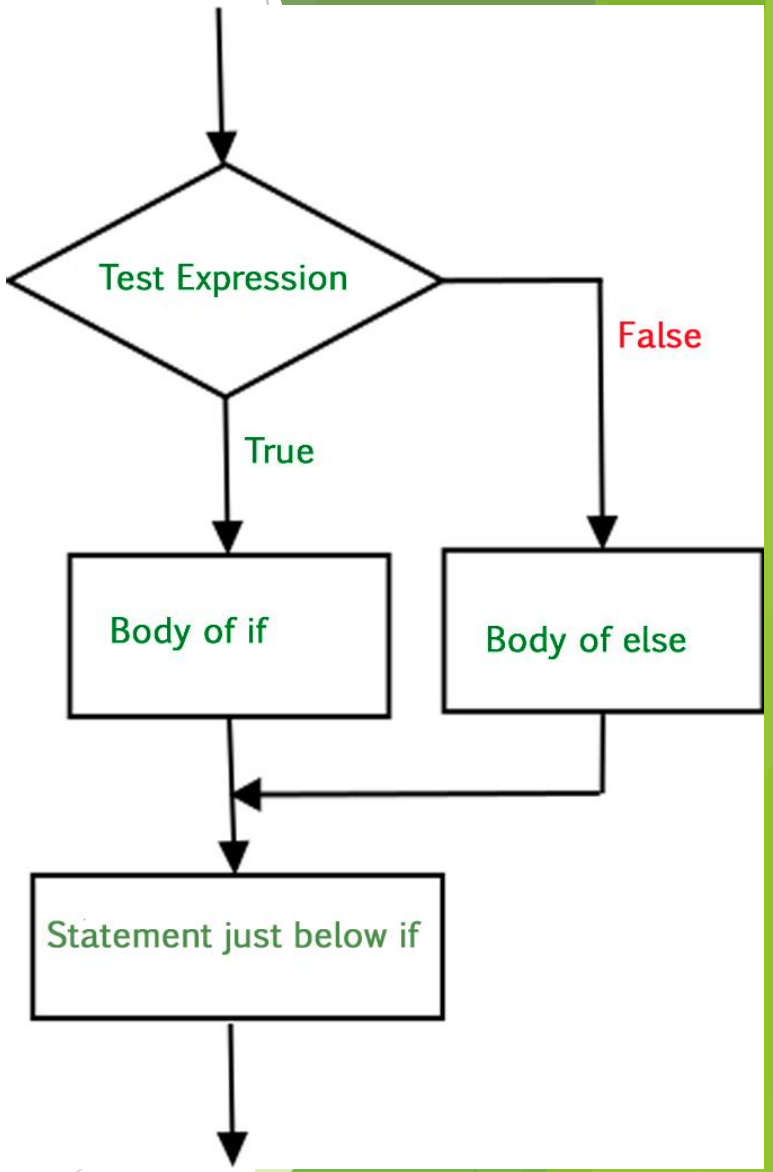
```
if a>b :
    print("A has more than B has")
```

If-

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

Example 1:

```
i = 2
if (i < 15):
    print ("i is smaller than 15")
else:
    print ("i is greater than 15")
```



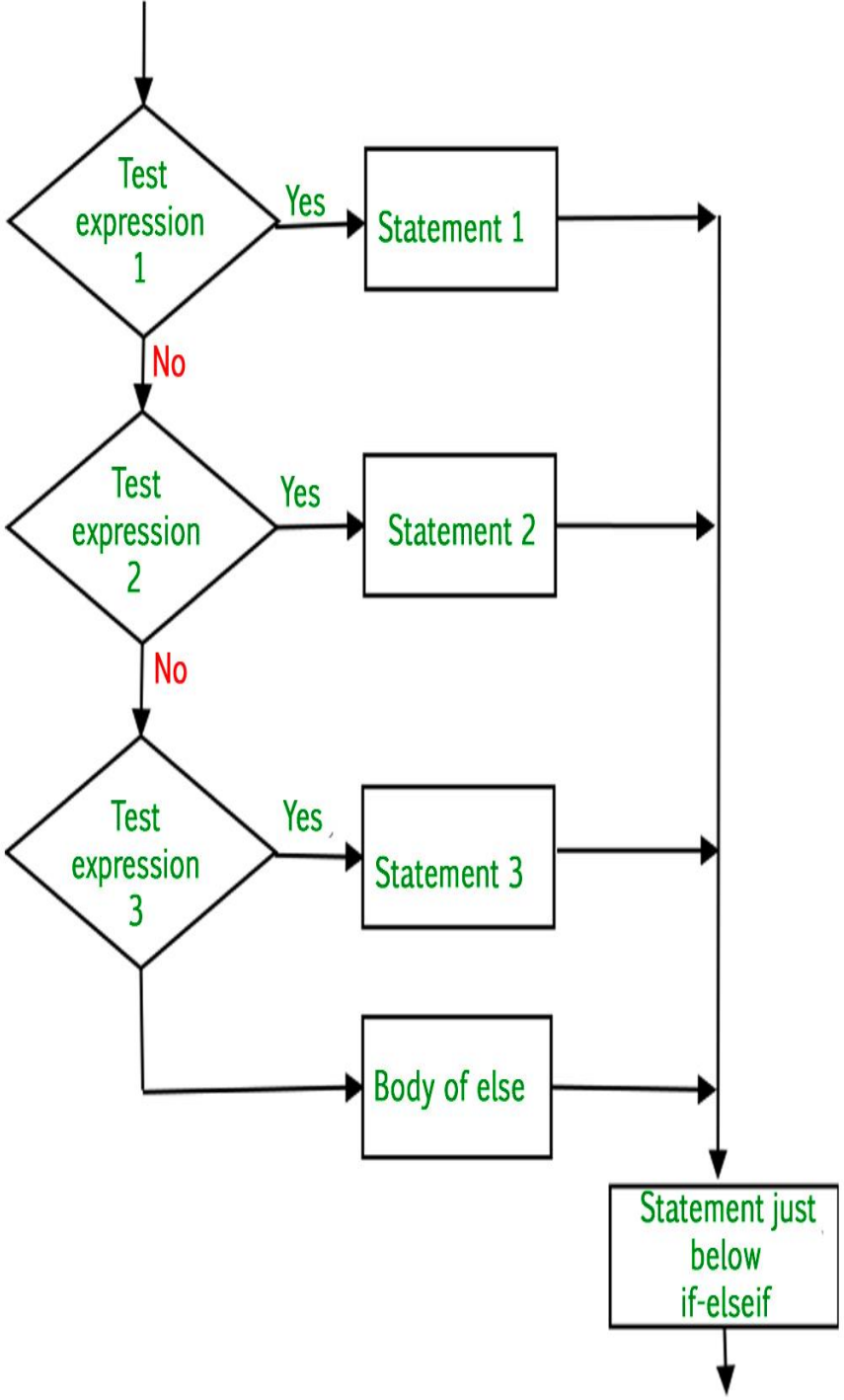
If-elif-else

Syntax :

```
if (condition):
        #statement
elif (condition):
        #statement
.
.
else:
        #statement
```

Example 1:

```
i = 20
if (i == 10):
        print ("i is 10")
elif (i == 15):
        print ("i is 15")
elif (i == 20):
        print ("i is 20")
else:
        print ("i is not present")
```



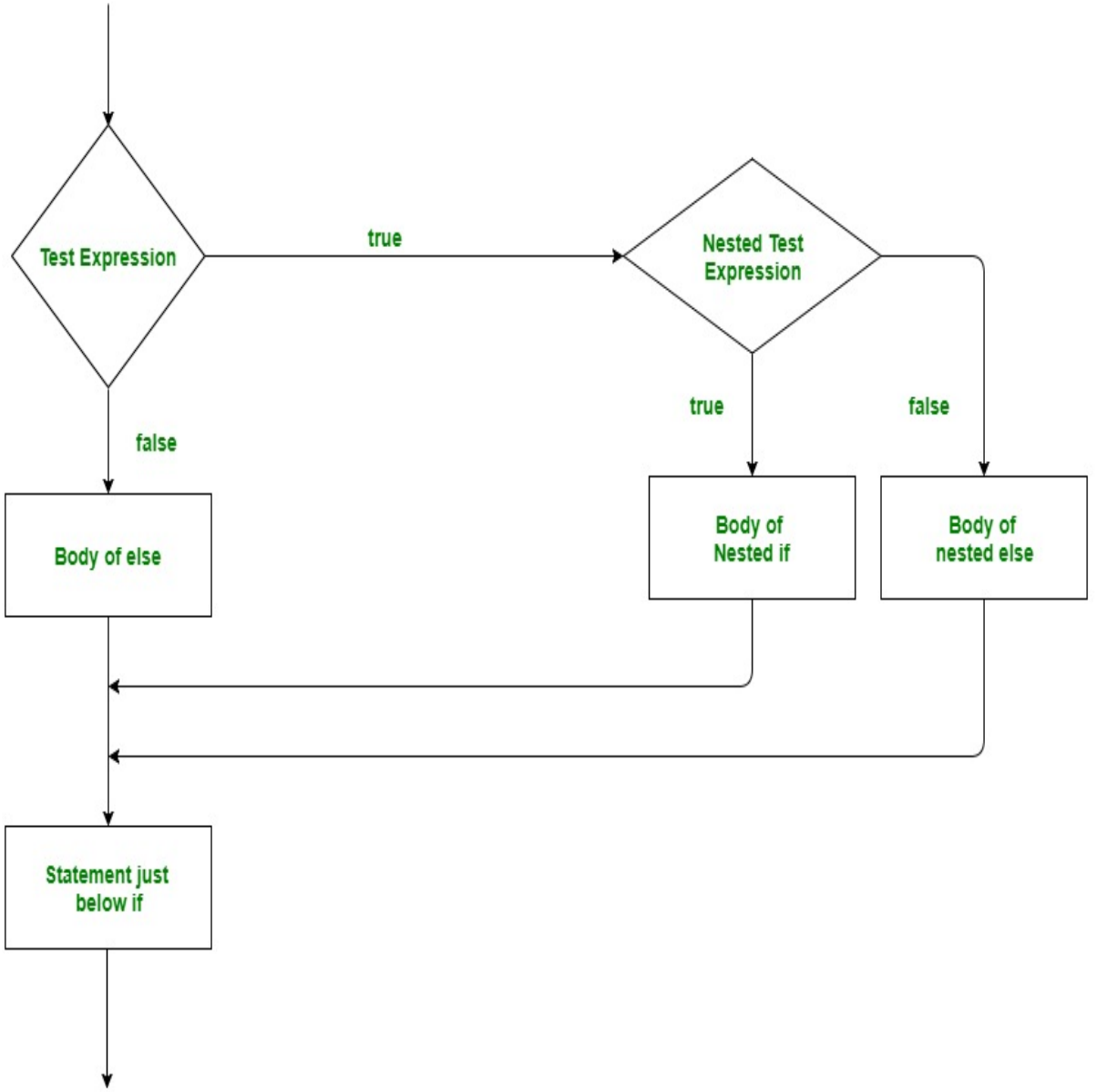
Nested if

Nested if statements means an if statement inside another if statement.

Syntax :

```
if (condition1):
    # Executes when condition1 is true
    if (condition2):
        # Executes when condition2 is true
        # if Block is end here
    # if Block is end here
```

```
num = float(input("Enter a number: "))
 if (num >= 0):
        if (num == 0):
                print("Zero")
        else:
                print("Positive number")
else:
        print("Negative number")
```



3. Repetition/Iteration

A **repetition statement** is used to repeat a group(block) of programming instructions until a certain condition fulfilled.
In Python, we generally have two loops/repetitive statements:

- for loop — COUNTING LOOPS

The loops that repeat a certain number of times

- while loop — CONDITIONAL LOOPS

The loops that repeat until a certain thing happens i.e the keep repeating as long as some condition is true

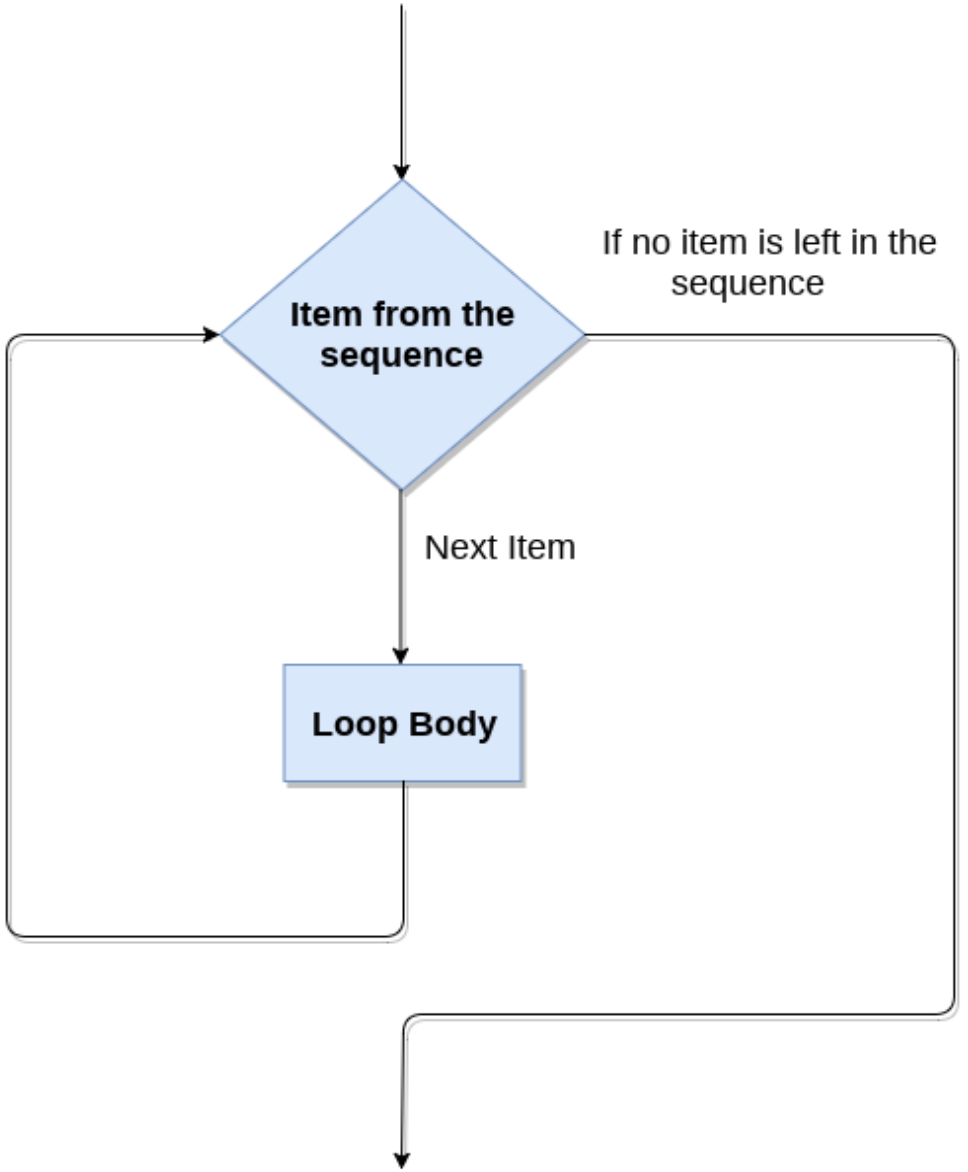
for LOOP

Each time , when the loop-body is executed ,is called an Itertion

Syntax:

```
for iterating_var in sequence:
    statements_to_repeat
```

```
str = "Python"
for i in str:
    print(i)
```



range( )

The **range()** function is used to generate the sequence of the numbers.

Syntax:

```
for iterating_var in range(start,end,step_size) :
        statements
```

```
for i in range(1,11,1):
    print(i)
```

- start: Starting number of the sequence.
- stop: Generate numbers up to, but not including this number.
- step(Optional): Determines the increment between each numbers in the sequence.

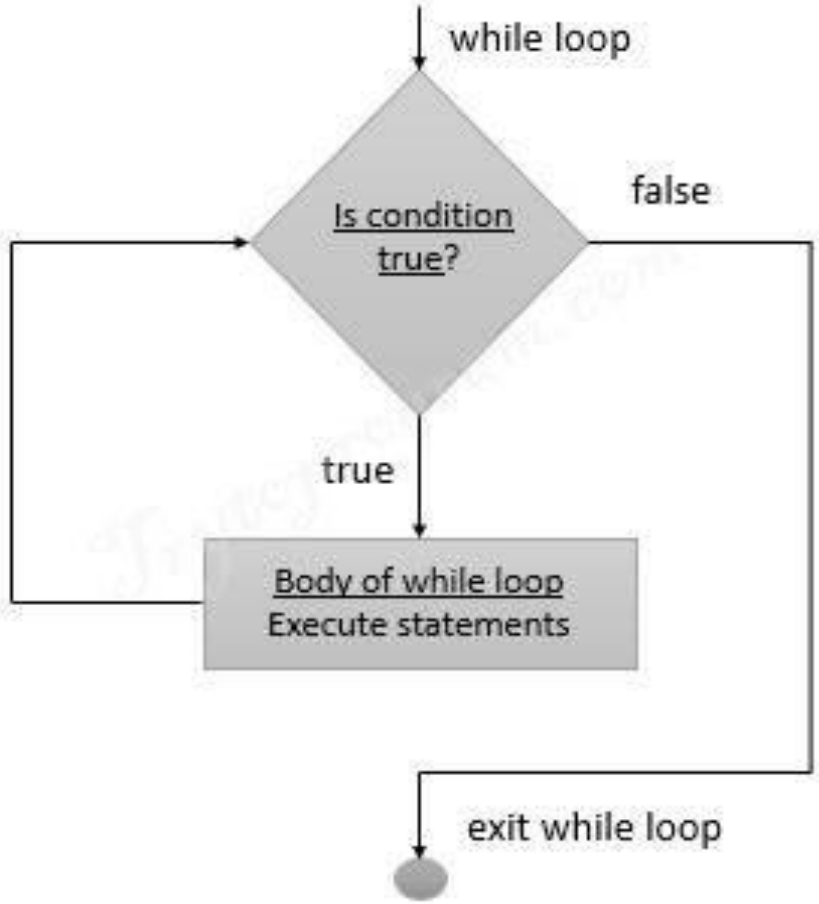
while LOOP

while loop keeps reiterating a block of code defined inside it until the desired condition is met.

Syntax:

```
while(condition):
    statement
```

```
i = 1
while (i < 10):
        print (i)
        i = i+1
```



Jump statements are

used to transfer the program's control from one location to another.

- Means these are used to alter the flow of a loop like - to skip a part of a loop or terminate a loop
- There are three types of jump statements used in python.

1.break
2.continue
3.pass

break statement

Terminates the loop statement and transfers execution to the statement immediately following the loop

```
for val in "string " :
    if val == "I" :
        break
        print(val)
print("The end")
```

continue statement

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

```
for val in "init" :
    if val == "i" :
        continue
        print(val)
print("The end")
```

pass statement

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

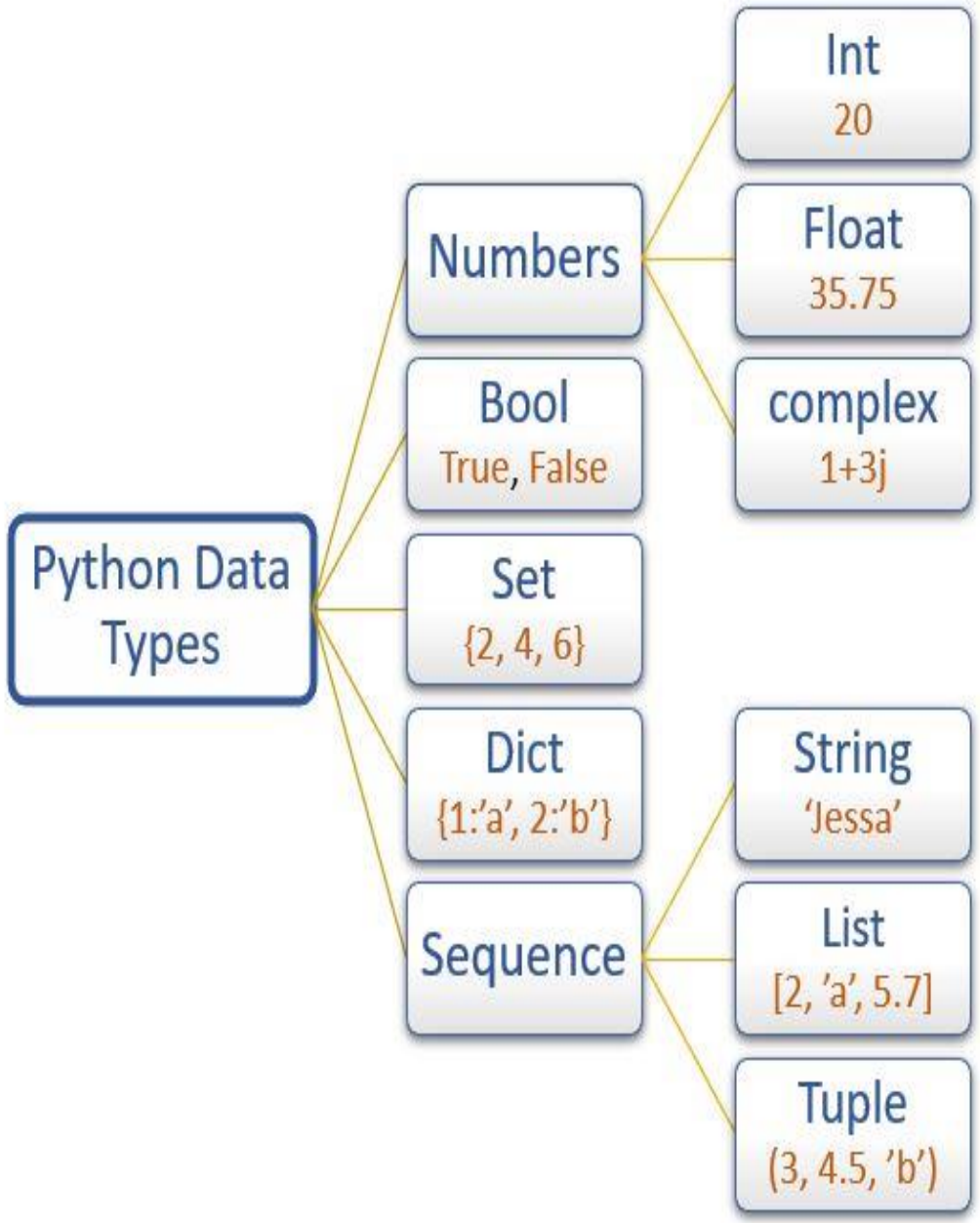
```
for i in 'initial’:
    if(i == 'i’):
        pass
    else:
        print(i)
```

DATA TYPE

A **DATA TYPE** IS A CHARACTERISTIC THAT TELLS THE INTERPRETER HOW A PROGRAMMER INTENDS TO USE THE DATA.

The data type of any object is found using the built in

`type()` function.



python data types can be broadly classified into two categories:

Immutable type
Mutable type.

An object whose internal state can be changed is called a mutable object

1.List
2.Set
3.Dictionary

An object whose internal state cannot be changed is called an immutable object.

1.Boolean
2.Numeric
3.String
4.Tuple

MANIPULATION ON

STRING
LIST
DICTIONARY

MANIPULATION ON

STRING

STRING

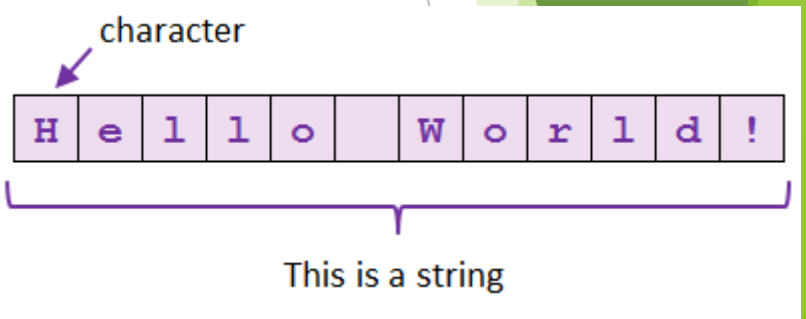
❑ Sequence of characters in

```
my_string = "This is a string"
my_string2 = 'This is also a string'
```

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

```
a = "Hello World!"
print(a)
```



String Length

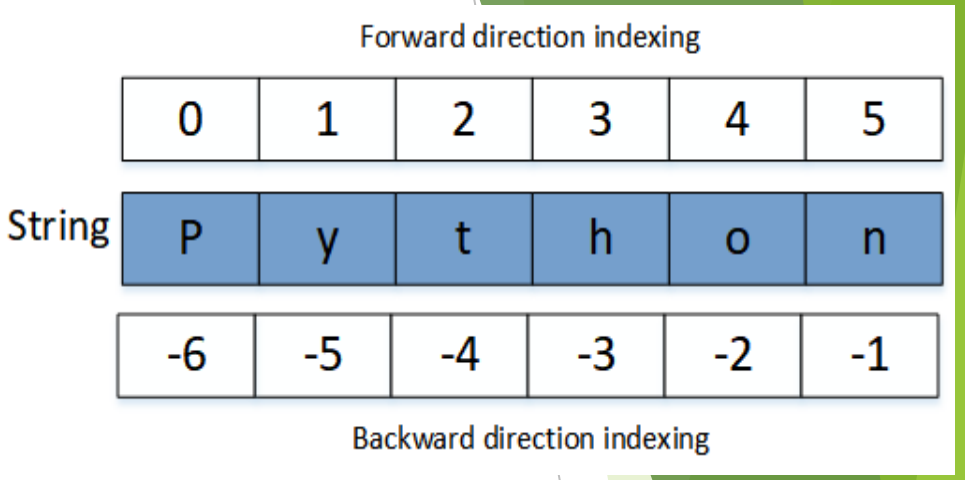
The length of a string represents the number of characters it contains.

To get the length of a string, use the `len()` function.

```
a = "Hello World!"
print(len(a))
```

POSITION OF CHARACTER IN STRING

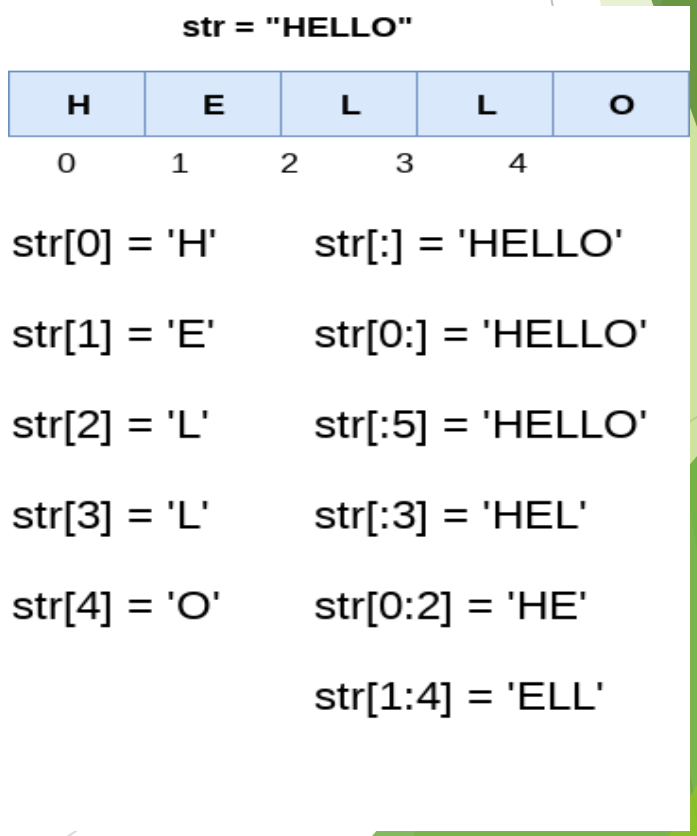
A character (also called element) of a string can be accessed with it's index number.



SlICING OF STRING

We can return a range of characters by using the slice syntax.

```
string[start:end:step]
```



TRAVERSING A STRING

Traversing refers to iterating through the elements of a string, one character each time.

```
name = “superb"
for ch in name:
    print(ch,‘-’,end=‘ ’)
```

Strings are immutable. This means that elements of a string cannot be changed once they have been assigned.

```
>>> my_string = ‘class11’
>>> my_string[5] = 'a’

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

String Special Operators

Traversing refers to iterating through the elements of a string, one character each time.

```
a="comp"
B="sc"
```

+	Concatenation - to add two strings	a + b comp
*	Replicate same string multiple times (Repetition)	a*2
[]	Character of the string	a[1] will
[:]	Range Slice –Range string	a[1:4] will
In	Membership check	p in a will
not in	Membership check for non availability	M not in a

will give 1

STRING METHOD and FUNCTION

```
stringObject.methodName()
```

Built-in String Functions in Python

Conversion Functions

capitalize() – Returns the string with the first character capitalized and rest of the characters in lower case.

lower() – Converts all the characters of the String to lowercase

upper() – Converts all the characters of the String to uppercase

swapcase() – Swaps the case of every character in the String means that lowercase characters got converted to uppercase and vice-versa.

title() – Returns the 'titlecased' version of String, which means that all words start with uppercase and the rest of the characters in words are in lowercase.

count(str[, beg [, end]]) – Returns the number of times substring 'str' occurs in the range [beg, end] if beg and end index are given else the search continues in full String Search is case-sensitive.

Comparison Functions

isalpha() – Returns ‘True’ if String contains at least one character (non-empty String), and all the characters are alphabetic, ‘False’ otherwise.

isupper() – Returns ‘True’ if all the characters in the String are in uppercase. If any of the char is in lowercase, it will return False.

islower() – Returns ‘True’ if all the characters in the String are in lowercase. If any of the char is in uppercase, it will return False.

isalnum() – Returns ‘True’ if String contains at least one character (non-empty String), and all the characters are either alphabetic or decimal digits, ‘False’ otherwise.

Comparison Functions

isalnum() – Returns ‘True’ if String contains at least one character (non-empty String), and all the characters are either alphabetic or decimal digits, ‘False’ otherwise.

isdigit() – this method returns True if the string contains only numeric digits(0 to 9) else returns False

isspace() – Returns ‘True’ if there are only whitespace character in the string.

Padding Functions

rjust(width,fillchar) – Returns string filled with input char while pushing the original content on the right side.
By default, the padding uses a space. Otherwise, ‘fillchar’ specifies the filler character.

ljust(width,fillchar) – Returns a padded version of String with the original String left-justified to a total of width columns
By default, the padding uses a space. Otherwise, ‘fillchar’ specifies the filler character.

center(width,fillchar) – Returns string filled with the input char while pushing the original content into the center.
By default, the padding uses a space. Otherwise, ‘fillchar’ specifies the filler character.

zfill(width) – Returns string filled with the original content padded on the left with zeros so that the total length of String becomes equal to the input size.
If there is a leading sign (+/-) present in the String, then with this function, padding starts after the symbol, not before it.

Search Functions

find(str [,i [,j]]) – Searches for ‘str’ in complete String (if i and j not defined) or in a sub-string of String (if i and j are defined).This function returns the index if ‘str’ is found else returns ‘-1’.

index(str[,i [,j]]) – This is same as ‘find’ method. The only difference is that it raises the ‘ValueError’ exception if ‘str’ doesn’t exist.

rfind(str[,i [,j]]) – This is same as find() just that this function returns the last index where ‘str’ is found. If ‘str’ is not found, it returns ‘-1’.

count(str[,i [,j]]) – Returns the number of occurrences of substring ‘str’ in the String. Searches for ‘str’ in the complete String (if i and j not defined) or in a sub-string of String (if i and j are defined).
Where: i=search starts from this index, j=search ends at this index.

String Substitution Functions

replace(old,new[,count]) – Replaces all the occurrences of substring ‘old’ with ‘new’ in the String.
If the count is available, then only ‘count’ number of occurrences of ‘old’ will be replaced with the ‘new’ var.
Where old =substring to replace, new =substring

split([sep[,maxsplit]]) – Returns a list of substring obtained after splitting the String with ‘sep’ as a delimiter.
Where, sep= delimiter, the default is space, maxsplit= number of splits to be done

splitlines(num) – Splits the String at line breaks and returns the list after removing the line breaks.
Where num = if this is a positive value. It indicates that line breaks will appear in the returned list.

join(seq) – Returns a String obtained after concatenating the sequence ‘seq’ with a delimiter string.
Where: the seq= sequence of elements to join

MISC String Functions

lstrip([chars]) – Returns a string after removing the characters from the beginning of the String.
Where: Chars=this is the character to be trimmed from the String.
The default is whitespace character.

rstrip() – Returns a string after removing the characters from the End of the String.
Where: Chars=this is the character to be trimmed from the String. The default is whitespace character.

rindex(str[,i [,j]]) – Searches for 'str' in the complete String (if i and j not defined) or in a sub-string of String (if i and j are defined). This function returns the last index where 'str' is available.
If 'str' is not there, then it raises a ValueError exception.
Where: i=search starts from this index, j=search ends at this index.

len(string) – Returns the length of given String

MANIPULATION ON

LIST

LIST

A list in Python is used to store the sequence of various types of data.

Python lists are mutable type its mean we can modify its element after it created.

```
thislist = ["apple", "banana", "cherry"]
```

It can have any number of items and they may be of different types (integer, float, string etc.).

Characteristics of Lists:

- **The lists are ordered.**
- **The element of the list can access by index.**
- **The lists are the mutable type.**
- **The lists are mutable types.**
- **A list can store the number of various elements.**

```
# empty list
my_list = []
 # list of integers
 my_list = [1, 2, 3]
# list with mixed data types
my_list = [1, "Hello", 3.4]
```

A list can also have another list as an item. This is called a **nested list.**

```
my_list = ["mouse", [8, 4, 6], ['a']]
```

Access Items in LIST

List items are indexed and you can access them by referring to the index number

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Range of Indexes

we can specify a range of indexes by specifying where to start and where to end the range.

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

Range of Negative Indexes

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

Basic List Operations

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

Python has a set of built-in methods that you can use on lists

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python List Built-in functions

cmp(list1, list2)	It compares the elements of both the lists.	This method is not used in the Python 3 and the above versions.
len(list)	It is used to calculate the length of the list.	L1 = [1,2,3,4,5,6,7,8] print(len(L1))
max(list)	It returns the maximum element of the list.	L1 = [12,34,26,48,72] print(max(L1)) 72
min(list)	It returns the minimum element of the list.	L1 = [12,34,26,48,72] print(min(L1)) 12
list(seq)	It converts any sequence to the list.	str = "Johnson" s = list(str) print(type(s)) <class list>

Dictionary

Python - 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. An empty dictionary without any items is written with just two curly braces, like this: {}.

keys must be of immutable type
(string, number or tuple with immutable elements) and must be unique.

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
```

Dictionary Items

Dictionary items are ordered, changeable, and does not allow duplicates.
Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

Access Dictionary Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

```
thisdict = { "brand":"Ford" , "model":"Mustang", "year":1964}

print(thisdict["brand"])
```

The `keys()` method will return a list of all the keys in the dictionary.

The `values()` method will return a list of all the values in the dictionary.

```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}

x = thisdict.values()
y= thisdict.keys()

print(x)
print(y)
```

Change Values

You can change the value of a specific item by referring to its key name:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Loop Through a Dictionary

```
thisdict = { "brand":"Ford" ,
"model":"Mustang", "year":1964}
```

There are multiple ways to iterate over a dictionary in Python.

- Iterate through all keys

```
for x in thisdict.keys():
  print(x)
```

- Iterate through all values

```
for x in thisdict.values():
  print(x)
```

- Iterate through all key, value pairs

Loop through both *keys* and *values*, by using the `items()` method

```
for x, y in thisdict.items():
  print(x, y)
```

Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary