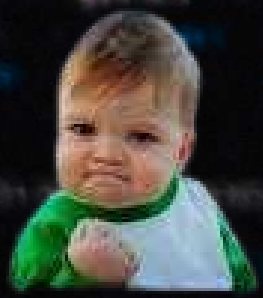
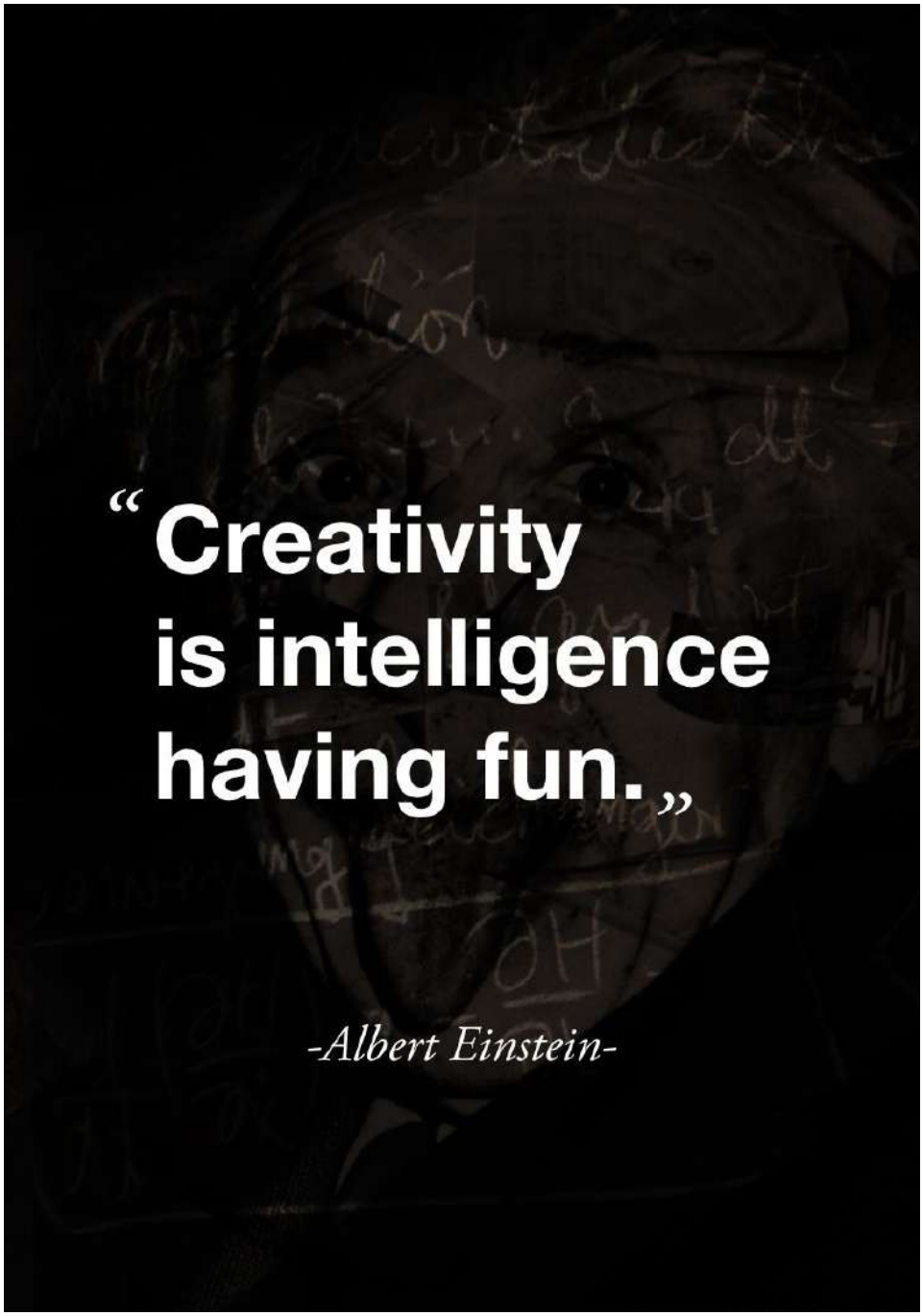
Can programmers have fun while learning ?

PYTHON THE MEME BOOK

PRIYAM KAPOOR




“Creativity
is intelligence
having fun.„
-Albert Einstein-

PREFACE

A SMALL NOTE

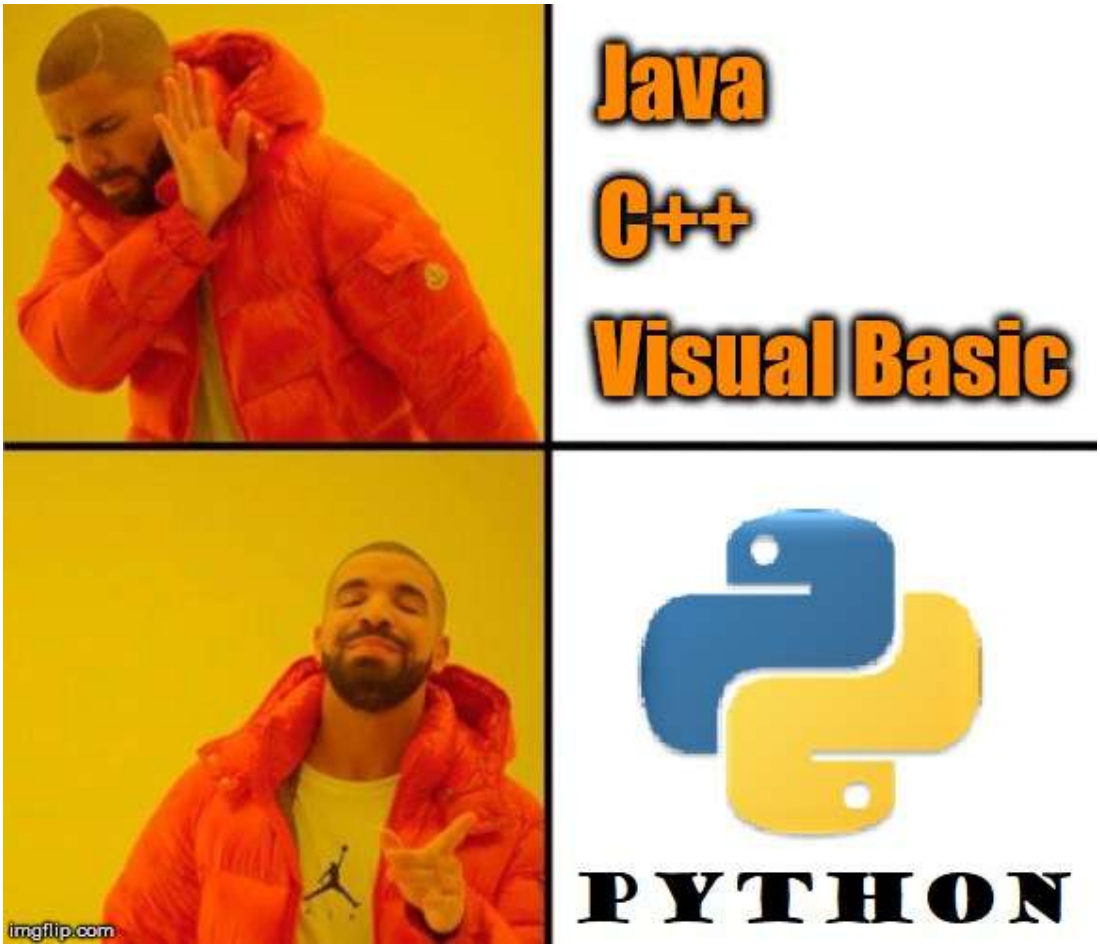
"Over the years, Python has gone through several stages of advancements and improvements and is still one of the popular coding language.
Anyone who's just embarked upon a journey of codes might relate to this book. I've compiled important concepts of python in relation with real world so as to instill practicality within the coder.
I've used memes, real-life instances, concept definition and multiple applicable programs for all the topics.
I would like to extend my sincere thanks to everyone who has motivated me and also helped me in compiling this book.

Hope it helps to learn python in a fun way."

Priyam Kapoor

PRIYAM KAPOOR

AUTHOR

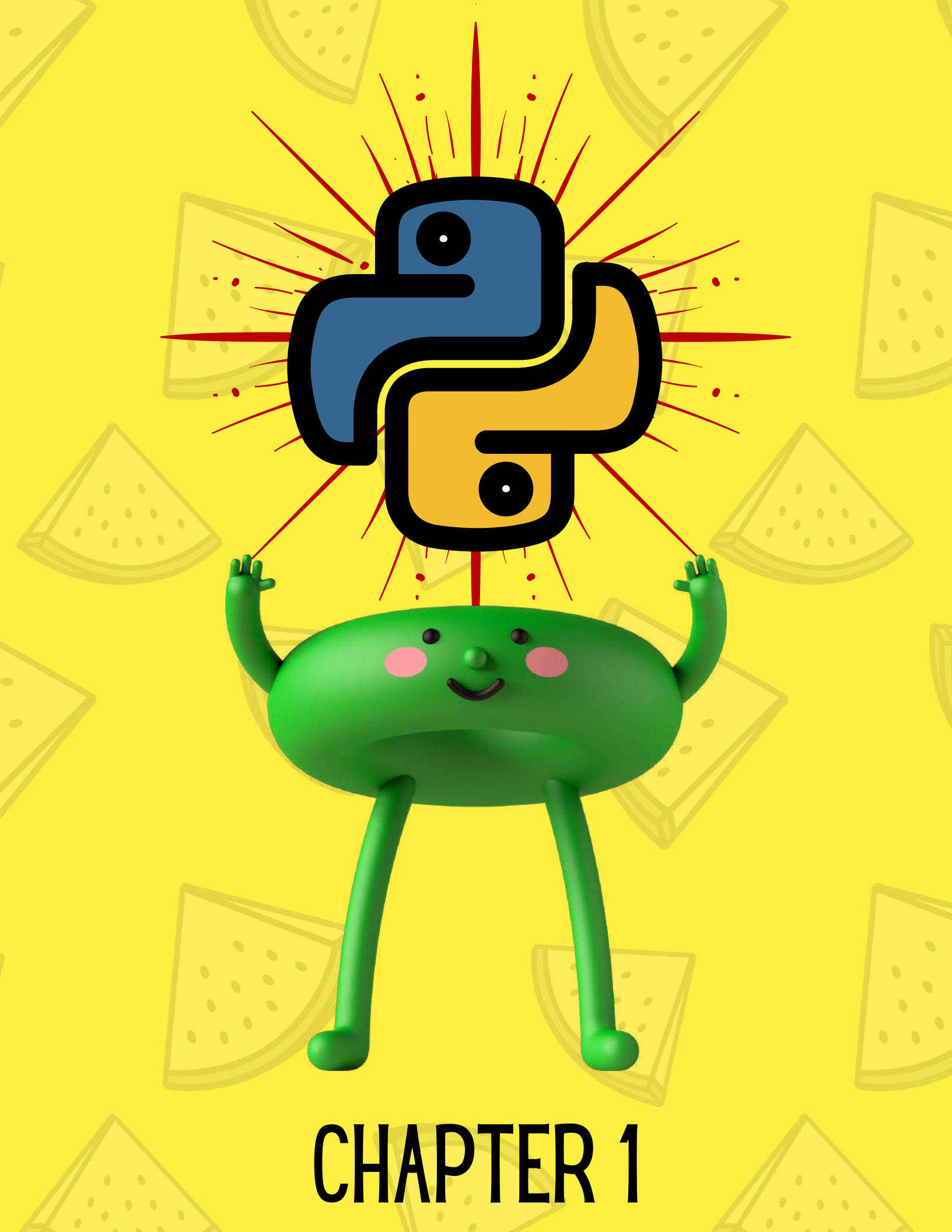


"ENJOY"

■ ■ ■

PYTHON 3.0
THE MEME BOOK

CONTENTS



CHAPTER 1

INTRODUCTION

Python: History as a programming language.

Python was built in the late 1980s by Guido Van Rossum in the Netherlands. Van Rossum is Python's principal author. Python was named for the BBC TV show Monty Python's Flying Circus.

Python 2.0 was Launched on October 16, 2000, with a shift to a more transparent and community-backed process.

Python 3.0, a big and backwards-incompatible release, was launched on December 3, 2008 after a long time of testing. This release marked a new beginning in the popularity of the language.

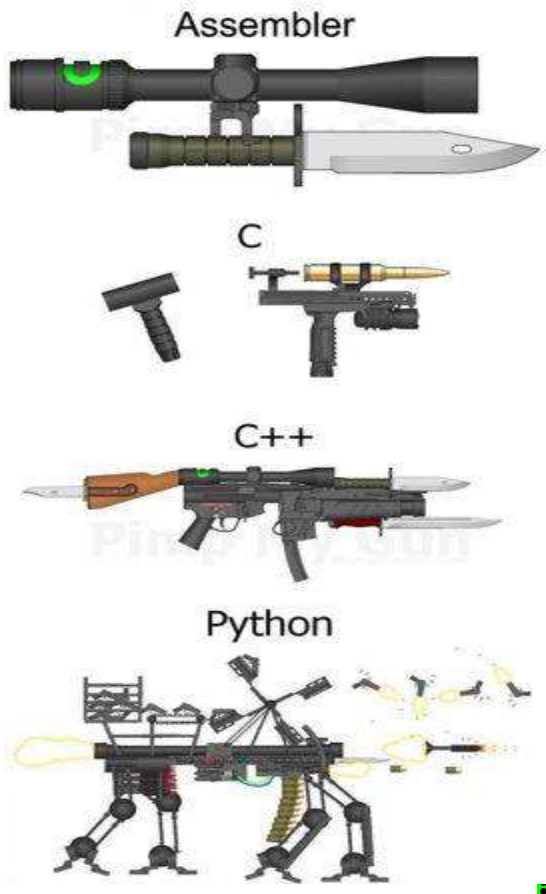
Advantages : Reasons of popularity

1. **One of the easiest to learn.**

Python is known for its English-Like commands, which makes it easier for people to understand and develop solutions in it.

2. **Large Community Support.**

After the 2008 release, Python 3.0 took the developer community by storm, many newbie programmers started to code in the language, forming a big base of modern and brilliant developers as Community.



3. Wide Applications Range

From Data science to Automation to Web development, python has its application everywhere.

Python brings advantages from both worlds. Most concepts in python are mixtures of two different concepts. Eg, list is a mixture of arrays and linked list.

One reason which many developers personally believe behind the popularity of python is that it hasn't reached its full efficiency as of now.

How can this be a reason for popularity? It can be. As we see with languages like java and C++, which were used extensively for years are more difficult to understand, with things and concepts they got loaded as the time flew.

So, we can say that this is the best time to learn Python.

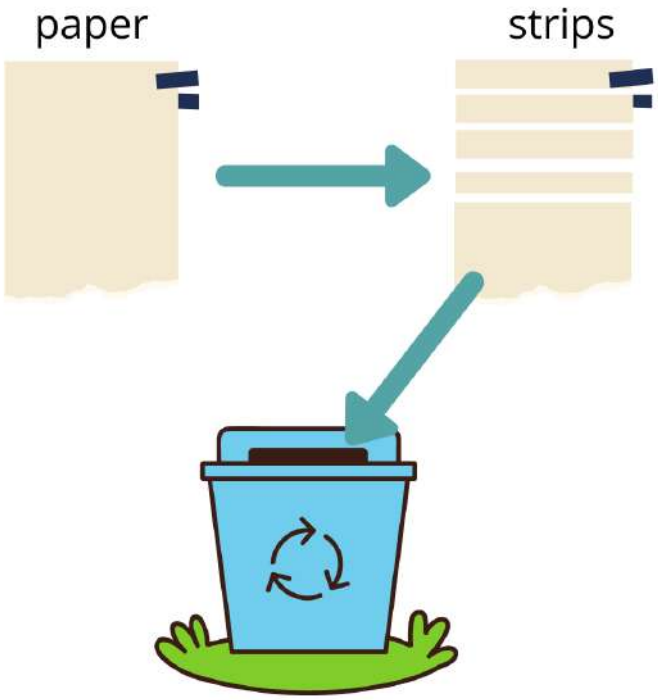
Disadvantage

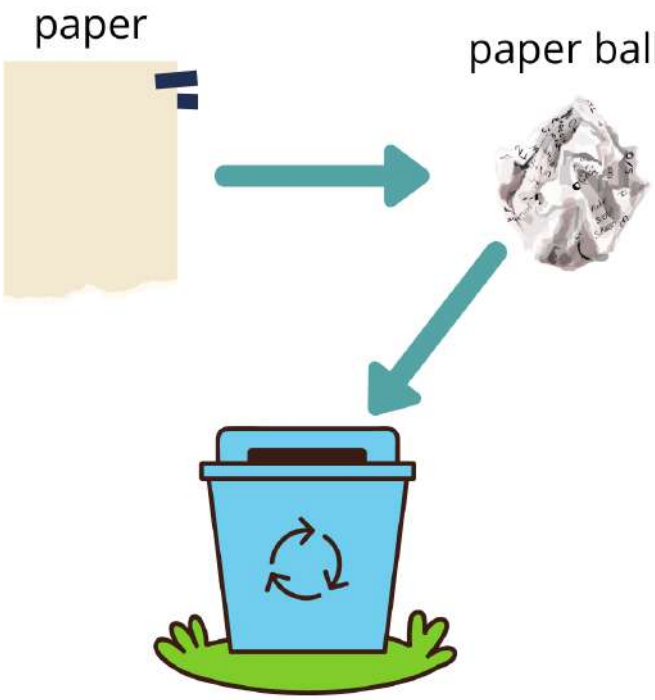
The biggest disadvantage of python is Speed. It is comparatively slower than many other languages.

Why is Python slower?

Python is slower because it is interpreted.

Interpreted	Compiled
When a code is executed Line by line it is called interpreted. Eg, if we have a paper and we cut a strip and throw it in trash one by one.	When a code is executed at once it is called compiled Eg. If we have a paper and we make a paper ball of it and throw it directly in the trash.





So it makes sense that interpretation is slower than compilation. Hence, Python is slower.

Installing Python On Your Machine

To code in python, we need a platform where we can write, run and debug our code. Many beginners stay confused in the options of softwares available for it and get more confused with the choices of other people.So, to make it easy let us compare the concept of python softwares with **pizza.** (Why? Because, who doesn't like pizza)

Recipe- Pizza

Distributors

As you can see in the above image, Pizza is a recipe. Everyone likes pizza. But it is distributed by many food companies. So, someone might like Dominos, someone else might like Pizza Hut. It's all on the choice of the particular person

Recipe- python.org

Distributers

Now, look at this image. Relate it with the above example. Python.org has the recipe of pizza while all the others are distributors of the language. So it basically doesn't matter much where you code on. What matters is where you can pull off your full potential.

Here are the steps to install normal python idle on your computer, others you can install on your own easily.

- Open Python.org
- Click on the Download tab.
- Select your Operating System.
- Select the desired version (preferably 3.6+ as of writing this book)
- Download the Installer
- Open Installer
- Check the Add to Path Option (it is important to make your python environmental variable)
- Install in default location.
- And it is installed.

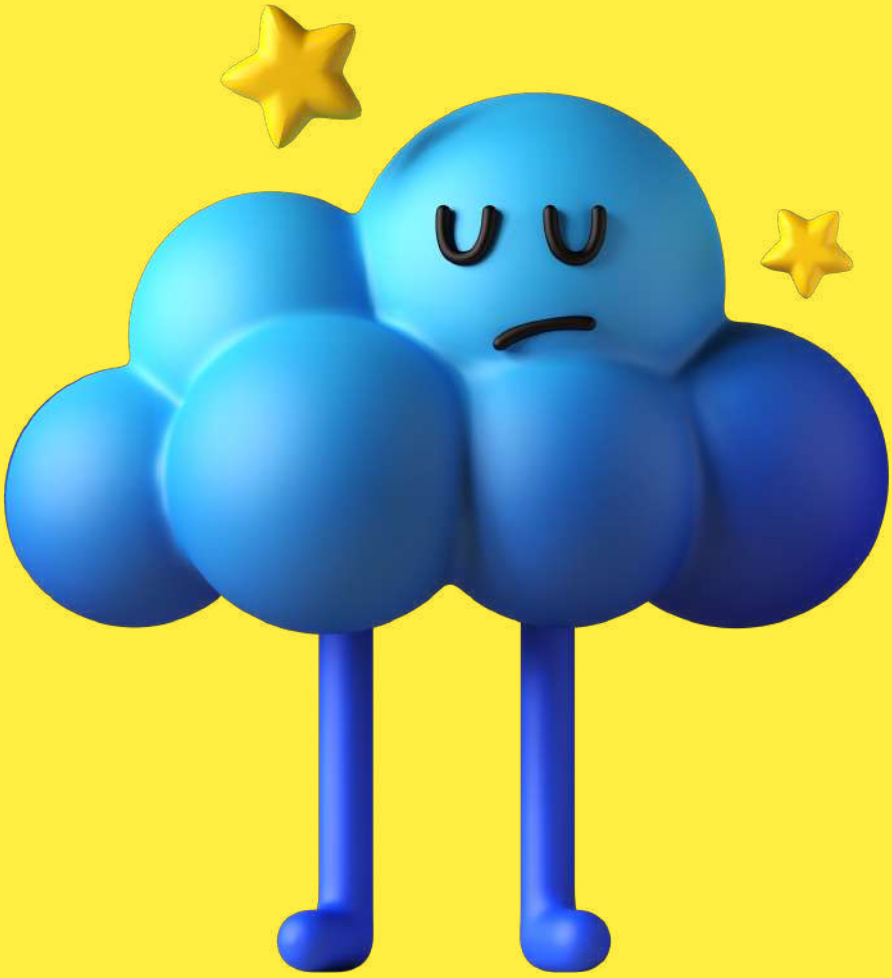
To check, whether it is correctly installed or not.
Open command prompt/ shell/ console
And type **python --version**, You will get the version of the python.

 You also get an editor with this **IDLE** . Use it to write your code.

You can install any other IDE or editor of your choice.

CHAPTER 2

BASICS OF PYTHON

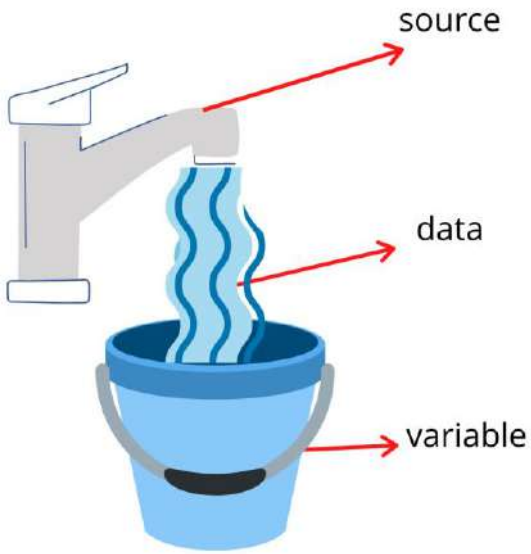


BASICS OF PYTHON

It is very important to create a strong base in any programming language to use it to its full potential.
In this chapter we will understand the basics of the Python Language.

Variables

Variables are named containers for storing the data.



Why do we need a variable?
Variables store the value for later use.
A person can remember the number 123, but remembering 213032439431481948104810 10 is very difficult. So we use variables for storing it.

In python, unlike other languages we do not need to declare a variable before using it.
We can directly use it.

We write
variable_name = Value to store

```
number = 1
```

```
a = 123
```

And just like there are some norms which we have to keep in mind while naming a person, we also have some **rules for naming a variable.**

- A variable name should not start with a number (they can be in middle or end).
- A variable name can not have a special character (not even space ‘ ’) but ‘_’ is allowed.
- A variable name can not be a Keyword (predefined meaningful words for python)
- Variables are case sensitive. Age is not equal to age.

```
1num = 123
```

```
  File "<ipython-input-4-4b625394839c>", line 1
    1num = 123
     ^
SyntaxError: invalid syntax
```

```
$num = 123
```

```
  File "<ipython-input-5-aac265453ea8>", line 1
    $num = 123
    ^
SyntaxError: invalid syntax
```

```
False =123
```

```
  File "<ipython-input-7-c3fa8ad98c3e>", line 1
    False =123
          ^
SyntaxError: can't assign to keyword
```

Data Types

Following the example of water as a data in the above topic, we can infer that water can be of different types depending on its uses (eg. tap-water, filtered-water), similarly data can also be of different types(eg. Numbers ,text). And these are known as data-types of a language.

TAP WATER

DRINKING WATER

Numbers

Text

Python provides three major data types.

1. **Numbers** - Numerical form of data.
   They are further divided into three sub categories :-
   - **int - numbers from -infinity to +infinity without a decimal point.**
     eg. -1, -2, 0, 4, 545354, 11
   - **float - numbers from -infinity to +infinity with a decimal point.**
     eg. -1.0, 2.99, 33.3333, 9.0, 0.0, 0.09
   - **complex - the numbers which have imaginary coefficient iota associated with them (** √-1 **= i** in mathematics and is displayed by **j** in Python**)**
     eg. 1+9.j, (1+8j)

2. **Strings** - Any sequence of characters enclosed within quotes are called Strings.
eg. ‘abcd’ , ‘1234’, “2python”, “““multi line”””.

Python quotes ->

- **‘’ (single quotes)** - normal string , no multi line allowed
- **“” (double quotes)** - normal string, no multi line allowed
- **“““””””” or ‘‘‘’’’ (triple quotes)** - strings with multi line feature

☆ **Why do we enclose a string in quotes?**

Ans.

If we take an example of a word **answer**, it follows all the three rules of variables, i.e., not starting with digit + no special characters + not a keyword.
So,python will treat it as a variable.
Hence, to differentiate strings from variables we keep them in quotes.

answer → will be considered as variable

“answer” or **‘answer’** → will be considered as String.

3. **Boolean -** Computers don’t answer directly yes or no to a question.
They have their own words for that.
These words are called boolean. And they are **True/False**

Eg. Is 5 equals to 5 ? **True**
Is 1>2 ? **False**

Operators

Operators are the symbols which perform a particular task.
In python, there are 7 types of operators, named :-

1. Assignment Operator
2. Arithmetic Operator
3. Relational Operator
4. Logical Operator
5. Identity Operator
6. Membership Operator
7. Bitwise Operator

We'll see them all ->

1. Assignment Operator

The **= (equals to)** is the assignment operator which assigns the value on the right
hand side to left side variable.

```
variable = 'value'

a = 10
b = 100
phone = 9140000000
name = 'Python'
```

2. Arithmetic Operator

The operators which performs mathematical functions are called Arithmetic Operator. They are 7 in number.

```
1. + (Addition) => adds two numbers
   3 + 7 => 10

2. - (Subtraction) => subtracts two numbers
   5 - 3 => 2

3. * (Multiplication) => multiplies two numbers
   11 * 5 => 55

4. / (float or point division) => gives quotient in float data type
   10 / 3 => 3.3333

5. // (Integer or floor division) => gives quotient in int data type
   10 // 3 => 3

6. % ( Remainder ) => gives remainder obtained
   10 % 3  => 1

7. ** (Exponent) => gives a raised to b
   2 ** 5 => 32
   3 ** 3 => 27
```

3. Relational Operator

The Operators which checks mathematical relations b/w two values.
They return boolean (True/False) answers.
These are ==, !=, >, <, <=, >=.

```
1. == (equals operator) --> checks equal

1 == 1
```

```
True
```

```
2. != ( not equals operator) --> checks not equal

1 != 1
```

```
False
```

```
3. >,< (greater and lesser than)
```

```
4. >=(greater or equals ) , <= (lesser or equals)
```

4. Logical Operators (or Conjunctions)

The logical Operators join two or more conditions.
They are **and, or, not.**

And

This operator results in **True** if all the conditions are correct otherwise **False**.

```
1==1 and 2==2
```

```
True
```

```
1>2 and 2==2
```

```
False
```

Or

This operator results in **True** if any condition is correct and **False** only if all are wrong.

```
1==1 or 2==2
```

```
True
```

```
1>2 or 2==2
```

```
True
```

```
1>2 or 2!=2
```

```
False
```

Not

This operator converts True statement to False and vice-versa.

```
not False
```

```
True
```

```
not True
```

```
False
```

```
1==1
```

```
True
```

```
not 1==1
```

```
False
```

5. Identity Operator

The identity Operator → **is**

This operator not only checks the equality b/w two values but also their data types and stored addresses too.

```
500 == 500.0
```

```
True
```

```
500 is 500.00
```

```
False
```

6. Membership Operator

The membership operator → **in**

This operator checks whether a particular element is in a collection or not.

```
'p' in 'python'
```

```
True
```

```
'k' in 'python'
```

```
False
```

```
Note --> case sensitive

'P' in 'python'
```

```
False
```

7. Bitwise Operator

The bitwise operators work on the Binary Digits of a number.
They convert the Integer (Base 10) to Binary(Base 2) and perform functions on it and again return to its updated Integer (Base 10).

```
1. >> Right shift ( m>>n)
   Shifts Number m (binary form) digits towards right by n steps.

5>>1
Binary of 5 -> 000000101
after shifting right -> 00000010 (which is 2 hence we get 2 as output)
```

```
2
```

```
2. << Left shift ( m<<n)
   Shifts Number m (binary form) digits towards left by n steps.

5<<1
Binary of 5 -> 000000101
after shifting left -> 00001010 (which is 10 in decimal hence we get 10 as output)
```

```
10
```

```
Other Bitwise Operators are -->

3. ~ (compliment)
4. & (bitwise and)
5. | (bitwise or)
6. ^ (bitwise xor)
```

The Output Command in Python

Real life Example	Python
	**print()**
We have this printer with us.What does a printer do? It takes our files or commands and prints it on a paper.	This here is a print() method to display the output on the console window of the python

```
print(value, ..., sep=' ', end='\n')

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments
```

```
print('hello world')
```

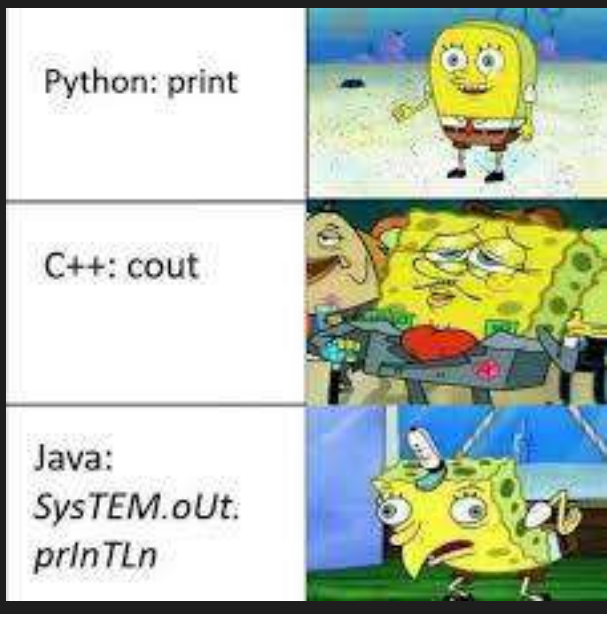
OUTPUT

```
>>> hello world
```

```
#can be separated by a , (by default gives a space)
print(1,'to','print')
```

OUTPUT

```
>>> 1 to print
```




```
#default value of separator can be changed by sep attribute
print(1,'to','print',sep='^^')
```

OUTPUT

```
>>> 1^^to^^print
```

```
#by default 2 print statements prints in different lines
#because they are ended with a new line
print(1)
print(2)
```

OUTPUT

```
>>> 1
    2
```

```
#default value of end can be changed by end attribute
print(1,end='?')
print(2,end='*')
```

OUTPUT

```
>>> 1?2*
```

```
#using sep and end together
print(1,2,3,sep='separator',end='end')
```

OUTPUT

```
>>> 1separator2separator3end
```

The Input Command in Python

If you want to make your snippet of code a general program, which means that can run for multiple user cases then you need to **take input from the user** of the data to check.

input()

So, for that we have input() Statement.

```
        #Syntax

        variable_name = input('message')

        #example

        name = input('please enter your name ')
        sub = input('please enter your favourite subject ')
print('Name is',name)
        print('Subject ',sub)
```

_____OUTPUT_____

```
>>> please enter your name Ashok
    please enter your favourite subject Python
```

```
>>>Name is Ashok
   Subject  Python
```

BY DEFAULT THE input() TAKES THE INPUT IN str (string) so we have to change the input with the help of convertors. The concept is called EXPLICIT TYPE CASTING.

```
        #taking input in integer
        # variable = int(input('message'))

        number = int(input('enter the number '))
```

_____OUTPUT_____

```
>>> enter the number 12345
```

```
#this won’t take words as input and will throw ValueError

number = int(input('enter the number '))
```

_____OUTPUT_____

```
>>> enter the number abc

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-c8feaee71088> in <module>
      1 #this won’t take words as input and will throw ValueError
      2
----> 3 number = int(input('enter the number '))

ValueError: invalid literal for int() with base 10: 'abc'
```

```
#taking input in integer
# variable = int(input('message'))

marks = float(input('enter the number '))
```

_____OUTPUT_____

```
>>> enter the number 78.9
```

```
#this won't take words as input and will throw ValueError
```

```
marks = float(input('enter the number '))
```

OUTPUT

```
>>> enter the number 78aaa
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-a47e78159024> in <module>
      1 #this won't take words as input and will throw ValueError
      2
----> 3 marks = float(input('enter the number '))

ValueError: could not convert string to float: '78aaa'
```

CHAPTER 3

FLOW CONTROL

FLOW CONTROL

LOOK AT THE BLOCK BELOW

```
print(1)
print(2)
```

You can not stop the second print() statement from printing 2 because it is the flow of the program.

We can control the flow by few methods

1) Conditionals -> if else elif
2) Loops -> for while
3) Functions
4) Jump Statements

Before going ahead let us see what does a statement means.

A **statement** is either **simple or compound**. A **simple statement** encloses no other **statement**. A **compound statement** can enclose **simple statements** and other **compound statements.**

SIMPLE STATEMENT

COMPLEX STATEMENT

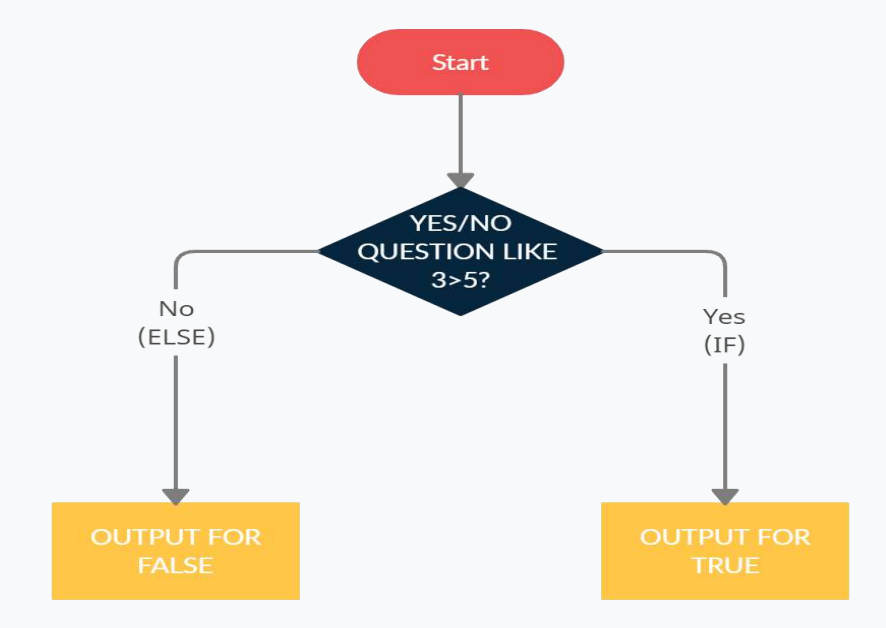
CONDITIONALS

A conditional is a statement which checks a particular statement for True or False and directs the flow of the program accordingly.

As we have already read that we have relational and logical operators in python so with the help of those and the conditional keywords **if, else, and elif.** You can control the flow of the code conditionally.

if - अगर (in hindi), तर (in marathi) , □ □ □ □(in kannada), എങ്കിൽ(in malayalam) and so on in multiple languages.

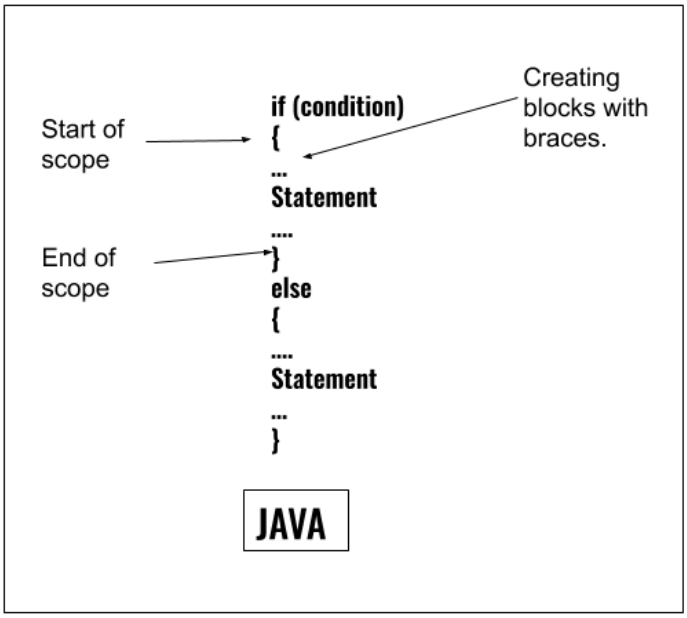
else - otherwise, □ □ □ □ □ (in hindi and marathi) , and so on in multiple languages.

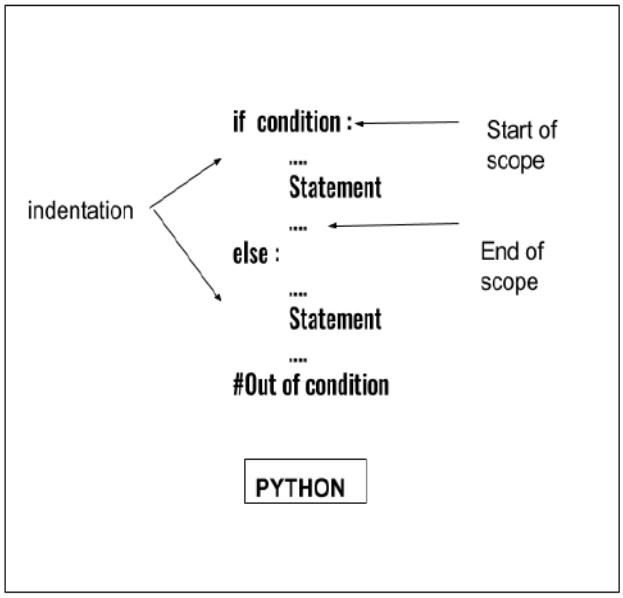



```
#SYNTAX
if condition :
    <
    statements
     >
else: #cannot be written without if
    <
    statements
     >
#out of if else statement
```

INDENTATION

Unlike languages like Java, C++, C, etc. which uses Curly Brackets {} to create dependent code blocks, Python uses indentation for creating dependent blocks of codes.





Python Programming

```
#Program of if-else
#age check for vote-eligibility

age = int(input('enter the age '))
if age>=18:
    print('Person Eligible to Vote')
else:
    print('Person is not Eligible to Vote')
```

OUTPUT

```
>>> enter the age 19
    Person Eligible to Vote
```

QUESTIONS TO TRY ON IF ELSE.

1) Checking whether Greatest among two numbers.
2) Checking whether a number entered is greater than 100 or not.
3) Check whether a number (n) divisible by another number (b) or not.

THE ELIF CLAUSE IN CONDITION

There are several questions which have multiple options unlike the yes/no question. So in that case we have to use the special elif clause.

```
#SYNTAX

  if condition 1:
      <
      statements
       >
  elif condition 2 if condition 1 fails:
      <
      statements
       >
  ...
  ...
  elif condition n if condition n-1 fails:
      <
      statements
       >
  else: # runs if all are false
      <
      statements
       >
  #out of if-elif-else ladder
```

It is like a ladder system. If the first condition is correct we execute the statement and stop but if it fails we are headed to check next and the same goes on for all elif conditions in chronological order and if all fails then and that moment the else part runs if present.

```
#example question
 # check which of the 2 number is  greatest or all are equal
 a = int(input('enter the number 1 '))
```

```
    b = int(input('enter the number 2 '))
    if a==b:
        print('Both of them are equal')
    elif a>b:
        print('A is greater')
    else:
        print('B is greater')
```

OUTPUT

```
>>> enter the number 1 19
    enter the number 2 19
    Both of them are equal
```

Questions with elif clauses.

Q1) Check the greatest number among 3 inputs.
Q2) Check if a year is leap year or not.
Q3) Build an easy basic calculator with python if-else ladder.

LOGICAL OPERATOR IN CONDITIONALS

We can use logical operators and, or, not in the conditions to check more than one conditions in a single if or elif statement.

```
#syntax

# and

if condition1 and condition2 and conditionN:
    <
```

```
        statement
        >

    #THIS RUNS THE STATEMENT IF ALL CONDITIONS ARE TRUE
```

```
    # or

    if condition1 or condition2 or conditionN:
        <
        statement
        >

    #THIS RUNS THE STATEMENT IF ANY CONDITION IS TRUE
```

```
    # not

    if not False_Condition:
        <
        statement
        >

    #CONVERTS FALSE CONDITION TRUE AND VICE VERSA
```

LOOPS

Just imagine that you have to repeat a certain line of code more than 10 times.

This task will be very time consuming because we will have to write the same code every time.

But we can use the concept of loops which are used to repeat certain lines of codes until a certain condition isn't met.

There are Two types of loops in Python :

1) for (membership loop)
2) while (conditional loop)

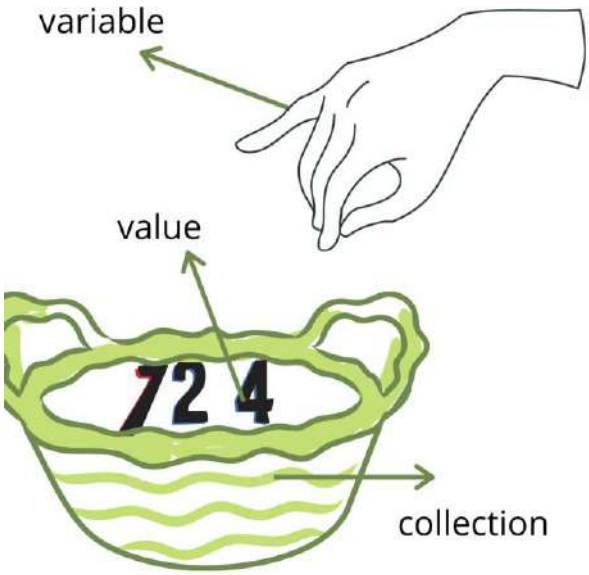
**** There is no do-while or switch case in python.**

FOR LOOP (MEMBERSHIP LOOP)

for loop is a membership loop. It takes the value out of a collection one by one.
Collection here can be String, range, list, tuple, etc.
for loop is helpful when we have to count a number range from start to stop or when we have to display the members of a collection.

```
#syntax of a for loop

for variable_name in collections :
    code here
```

```
# variable name ---> i
# collection ----> 'python'

for i in 'python':
```



```
        print('this is',i)
```

_____OUTPUT_____

```
>>> this is p
    this is y
    this is t
    this is h
    this is o
    this is n
```

The working of the above code is that.

Step 1.

i becomes ‘p’

And it gives output as → **this is p**

Step 2.

i becomes ‘y’

And it gives output as → **this is y**

Step 3.

i becomes ‘t’

And it gives output as → **this is t**

Step 4.

i becomes ‘h’

And it gives output as → **this is h**

Step 5.

i becomes ‘o’

And it gives output as → **this is o**

Step 6. **i** becomes ‘n’

And it gives output as → **this is n**

FOR LOOP AS A COUNTING LOOP

There is this special method named **range(), which returns the integers from one point to another with a default difference of 1.**

```
#range function

  range(stop) -> range object
  range(start, stop[, step]) -> range object

  it returns a closed packet of integers which can
  be accessed through for loop.

  #the Stop is exclusive

  range(1,10)
```

_____OUTPUT_____

```
>>> range(1, 10)
```

```
  range(1,10,2)
```

_____OUTPUT_____

```
>>> range(1, 10, 2)
```

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

```
>>> 1
    2
    3
    4
    5
    6
    7
    8
    9
```

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

_____

_____OUTPUT_____

```
>>> 1
    3
    5
    7
    9
```

Question: Print the Even numbers from range n to m.

Solution:

```
n = int(input('enter the start number ')) #start range
m = int(input('enter the stop number ')) #stop range

for i in range(n,m): # unpacking values from range() object
   if i%2==0: # condition for even numbers
       print(i) #printing the numbers
```

_____

_____OUTPUT_____

```
enter the start number 1
enter the stop number 20
2
4
6
8
10
12
14
16
18
```

QUESTIONS BASED ON FOR LOOP.

Q1) Take a number from the user and print its factorial.

Factorial of 5 → 1*2*3*4*5 = 120

Q2) Display the sum of series
1-2+3-4+5-6+7-8+9……..n terms

WHILE LOOP (CONDITIONAL LOOP)

while loop is a conditional loop. It repeats the code block while a certain condition is satisfied. We can use while loop when the condition sounds like " this has to run until some condition".

```
#syntax of while loop

while <condition>:
    ...
```

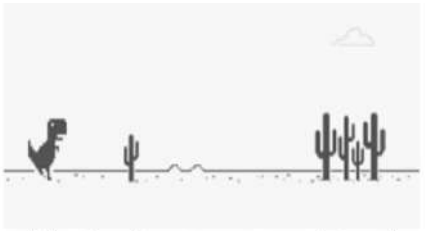
```
        code#this will run if condition is true
        ...
        #end of code it goes back up again to check for the condition


    #example
    i = 1 #initialization
    while i<5: # while condition
        print(i) # code
        i = i+1  # updating condition
```

OUTPUT

```
>>> 1
    2
    3
    4
```

Explanation:

Step 1. Initially i is 1
Step 2. Then i is checked whether it is lesser than 5 or not
Step 3. Since i is lesser than 5 so print(i) gives 1 as output
Step 4. then i is updated from 1 to 2
Step 5. Then again it is checked at the while condition
And this goes on till i is not equal to 5.



while the dino is not touching the cactus.

Keep Running the game

Guessing Number Game with While loop

Steps :

1. Take a random number in a variable
2. Take a number from the user
3. Check whether the entered number is random or 0 to quit.
4. If not then ask for number again inside the while loop
5. Else go out and check with if-else
6. If the number is random number then print you won
7. Else print that you lost.

- Point to note. The while loop will only end if entered number is either the random number or 0.

#code

```
 r = 19
#we are taking number on our own later we will random library
 n = int(input('guess a number')) # asking user to guess
 while n!=r or n==0: # while the guessed number not equal to origina
       n = int(input('Guess again or enter 0 to quit'))#updation
       #user might give a number to continue game or press 0 to quit
 if n==r:
```

```
        print('you won')
    else:
        print('you quit')
```

OUTPUT

```
>>> guess a number1
    Guess again or enter 0 to quit4
    Guess again or enter 0 to quit5
    Guess again or enter 0 to quit19
    you won
```

Else clause in loops

Just like an if ,with the else statement we can run a block of code once when the condition\ or collection no longer is true or left respectively in loops too.

```
while <condition> :
    #this runs while condition is true
else:
    #this runs if the condition is false

OR WITH FOR

for i in range():
    #code
else:
    #when collection is iterated

i = 1
while i != 5:
    print(i,'while')
```

```
        i=i+1
    else:
        print(i,'else')
```

OUTPUT

```
>>> 1 while
    2 while
    3 while
    4 while
    5 else
```

```
    for i in range(0,3):
        print(i,'for')
    else:
        print(i,'else')
```

OUTPUT

```
>>> 0 for
    1 for
    2 for
    2 else
```

Jump Statements → break and continue

- break is a keyword which stops the execution of the loop and throws the flow out of the loop.

```
    i = 1
    while i!=10:
        if i==5:# when i is 5
            break# break stops the execution of loop and send out
```

```
        print(i)
        i=i+1
    print('end')
```

OUTPUT

```
>>> 1
    2
    3
    4
    end
```

- continue is a keyword which sends the execution of the loop to checking condition again skipping the further part

```
    i=0
    while i!=9:
        i=i+1
        if i==5:
            continue# it will send execution again to while so 5 is not printed
        print(i)
```

OUTPUT

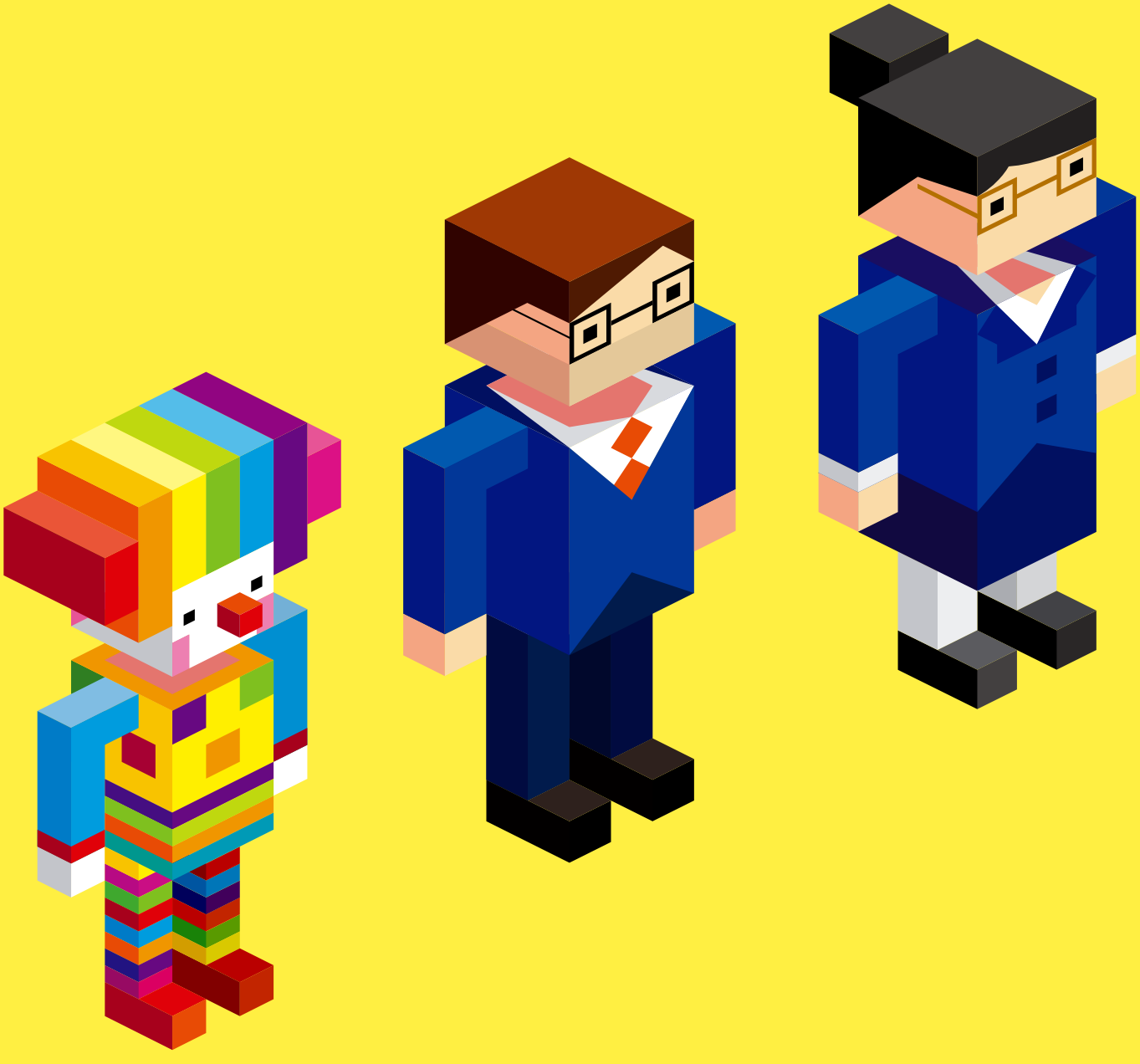
```
>>> 1
    2
    3
    4
    6  #missing 5
    7
    8
```

QUESTIONS:

1) Taking numbers from users until they don't press 'q' to quit. Display sum at last.
2) Print a factorial of a number. 5! = 1*2*3*4*5 = 120

CHAPTER 4

STRINGS



STRINGS

Any collection of characters enclosed within quotations is called strings.

We use quotations to differentiate strings from variables and whatsoever we give in quotes and print, they are printed the exact same.

“Anything inside quotes”

```
print('Hello world')
```

OUTPUT

```
>>> Hello world
```

The Types of quotes which we can use in strings are (with their use condition):

- **‘’ (single quotes)** - normal string , no multi line allowed
- **“” (double quotes)** - normal string, no multi line allowed
- **“““””””” or ‘‘‘’’’ (triple quotes)** - strings with multi line feature

Working with Strings

Calculating length of a String (Number of character)

To get the length of the string we use a **len()** function.

```
a = 'python is great'
print('The length is', len(a))
```

OUTPUT

```
>>> The length is 15
```

- It counts each and every character, even the space.

Indexing (POSITIONING OF CHARACTER IN STRINGS)

The strings are indexed in two ways in python.

1. LEFT TO RIGHT (Positive Indexing)
   From 0 to len(string)-1.

2. RIGHT TO LEFT (Negative Indexing)
   Starting right at -1 to -(len(s)+1)

```
a = 'python'

print(a[0])
print(a[1])
print(a[2])
print(a[3])
print(a[4])
print(a[5])
```

OUTPUT

```
>>> p
```

```
    y
    t
    h
    o
    n
```

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

OUTPUT

```
>>> n
    o
    h
    t
    y
    p
```

```
        #DON'T GIVE INDEXES OUT OF RANGE OR YOU'LL GET INDEXERROR
        k ='abc'
        print(k[100])
```

OUTPUT

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-18-e9ba20d063f7> in <module>
      1 #DON'T GIVE INDEXES OUT OF RANGE OR YOU'LL GET INDEXERROR
      2 k ='abc'
----> 3 print(k[100])

IndexError: string index out of range
```

STRINGS ARE IMMUTABLE

If a particular data value cannot remove, add or replace its content, that data type is known as immutable.

```
s = 'abcd'
print(s[1])
```

OUTPUT

```
>>> b
```

This is normal indexing and we get the value.
Lets try changing this 1 indexed value (‘b’) to ‘z’.

```
s[1] = 'z'
```

OUTPUT

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-f9407f874b96> in <module>
----> 1 s[1] = 'z'

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

We get an error, which states that str (string) doesn’t support item replacing.
Thus, supporting the fact that **strings are immutable.**

SLICING OF STRINGS

Cutting the required part of a string using indexes is known as Slicing.

NORMAL SLICING

We can see that indexing is helpful when we want only one character. But it turns out to be tedious if we want characters from one index to another.
This can be solved by using indexes.

Variable[Start_index : Stop_index]

stop_index is a wall. Which means that it is exclusive and that index is not added in the sliced part of string. (refer image example to understand clearly).

```
s = 'python'
```

The above is our original string s.

Let's say I want characters from 1 to 3 of this string.

```
print(s[1]+s[2]+s[3])
```

_____OUTPUT_____

```
>>> yth
```

This is how we can get multiple values using normal indexes, but as you can see that it is a very time consuming task.

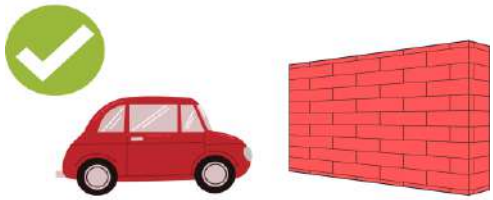
So, for the same we use Slicing.

Steps to do normal slicing.

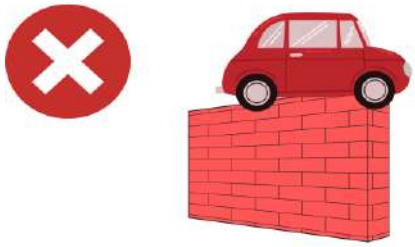
1. Select a start point
2. Mention colon symbol **(:)**
3. Enter the stop point, but remember that it should be at least one greater than the original stop **(THE WALL CONCEPT : one stops before the wall not on the wall)**
4. You'll get desired output after giving it a run.

   START == 1

   STOP == 4 (WILL MAKE IT STOP AT 3)



a car stops before the wall



not on the wall

```
print(s[1:4])
```

_____

_____OUTPUT_____

```
>>> yth
```

Now let's see what if we don't follow the WALL CONCEPT.

START == 1

STOP == 3 (WILL MAKE IT STOP AT 2)

```
print(s[1:3])
```

_____

_____OUTPUT_____

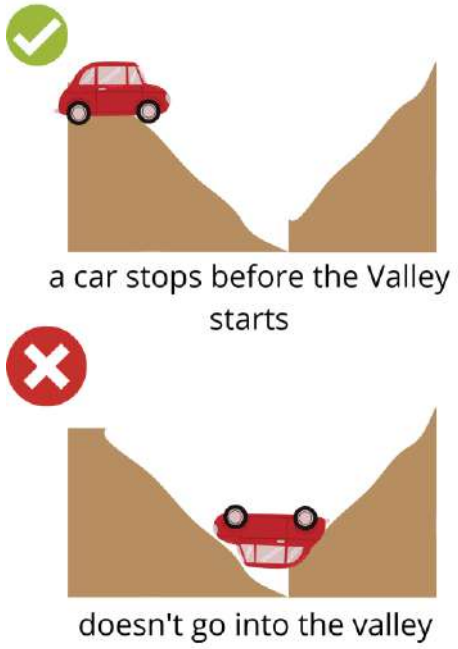
```
>>> yt
```

Now what if we give something out of the range of length of the string as stop.

Here, We will see the VALLEY CONCEPT.

START == 0

STOP == 100 (WILL MAKE IT STOP AT len(s))




```
print(s[0:100])
```

_____OUTPUT_____

```
>>> python
```

SLICING WITH STEPS

Variable[Start_index : Stop_index]

The above displayed syntax is known to us. It is normal indexing.

Basically what it means, it returns every character from start till stop-1 and that too **1 by 1.**

It is like walking up stairs **1 step each.**

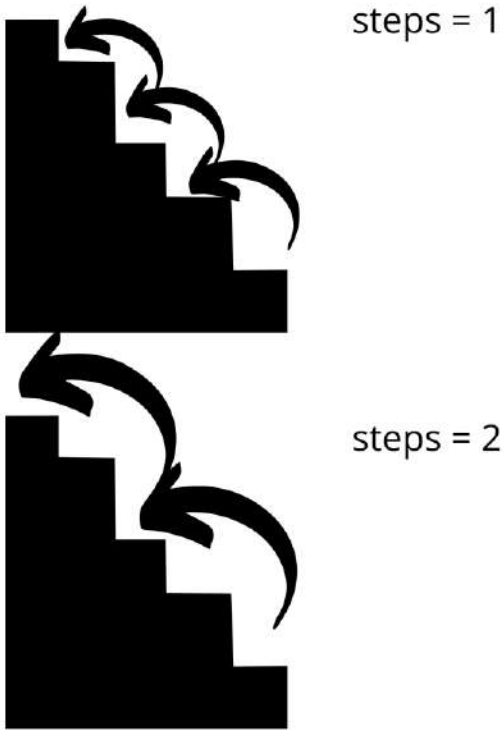
There is a way to change this number of steps of slicing.

Let's say You want all the even characters from start till end.

It can be Start from **0**, end at **length of string** and steps **2** every time.

So, writing this in syntax is as follows.

Variable[Start_index : Stop_index **: Steps**]



By Default Step is 1.

```
s = 'python'
```

This is the String which we have with us.
Task is to print every second character.

```
print(s[0:len(s):2])
```

_____OUTPUT_____

```
>>> pto
```

This is the method where we can use steps as **2**, to get every second character in the string.

```
print(s[0:len(s):3])
```

OUTPUT

```
>>> ph
```

This is the method where we can use steps as **3**, to get every second character in the string.

SLICING WITH NEGATIVE STEPS

If we mention the steps as -1, then it goes RHS to LHS so we will have to give appropriate indexes for the same.

```
print(s[5:1:-1])
```

OUTPUT

```
>>> noht
```

Using Positive indexes with Negative steps,

```
print(s[-1:-4:-1])
```

OUTPUT

```
>>> noh
```

Using Negative indexes with Negative steps,

DEFAULT VALUES OF SLICING PARAMETERS

By default, if we don't mention the parameters they get particular values loaded.

Start → if we don't mention start then it becomes 0.

Stop → if we don't mention stop then it becomes length of the collection. len(collection)

Step → if we don't mention step then it becomes 1.

```
s = 'python'
```

This is the String which we have with us.
Task is to show default values of the parameters.

```
print(s[0:len(s):1])
```

_____

_____OUTPUT_____

```
>>> python
```

These values are the default values.

```
print(s[::])
```

_____

_____OUTPUT_____

```
>>> python
```

If we don't mention them we get the same result.
It can be used singularly as well.

SPECIAL CASE

If we mention Step as -1 then the default value of Start and Stop changes.

When Steps = -1

Start → -1

Stop → - (len(collection)+1)

```
        print(s[-1:-(len(s)+1):-1])
```

_____OUTPUT_____

```
>>> nohtyp
```

These values are the default values.

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

_____OUTPUT_____

```
>>> nohtyp
```

If we don't Start and stop while step is -1 we get the same result.

USING SLICING TO **REVERSE THE STRING**

We can reverse a string using slicing with negative steps.

```
s = input('enter any sentence ')
print(s[::-1]) # reverse
```

_____OUTPUT_____

```
>>> enter any sentence World is One
    enO si dlrow
```

```
s = input('enter any sentence ')
print(s[::-1]) # reverse
```

_____OUTPUT_____

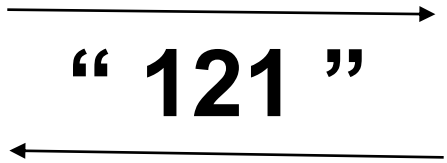
```
>>> enter any sentence programmer
    remmargorp
```

```
s = input('enter any sentence ')
print(s[::-1]) # reverse
```

_____OUTPUT_____

```
>>> enter any sentence 1213334
    4333121
```

Palindrome Program with Strings and Slicing



FROM BOTH THE SIDES
READS THE SAME

Step 1. Take input of string
Step 2. Save the reverse using slicing in a variable
Step 3. Compare the Original with the sliced variable string.
Step 4. If equal then its Palindrome
Step 5. Else it is not palindrome.

```
word = input('enter the word or number you want to check for palindrome ')
rev_word = word[::-1]

if word == rev_word:
   print('PALINDROME')
else:
   print('NOT PALINDROME')
```

```
OUTPUT 1→
>>> enter the word or number you want to check for palindrome malayalam
    PALINDROME

OUTPUT 2 →
>>> enter the word or number you want to check for palindrome python pro
    NOT PALINDROME
```

Methods of Strings

Though, Strings are immutable but still we can use methods to get a copy of the updated value of the same.
Here is a list of available methods of string and their description.

It’s advised that you try them all on your own and see how they work to remember them for the long term.

They are very handy.

	Method	Description
•	capitalize()	Converts the first character to upper case
•	casefold()	Converts string into lower case
•	center()	Returns a centered string
•	count()	Returns the number of times a specified value occurs in a string
•	encode()	Returns an encoded version of the string
•	endswith()	Returns true if the string ends with the specified value
•	expandtabs()	Sets the tab size of the string

•	find()	Searches the string for a specified value and returns the position of where it was found
•	format()	Formats specified values in a string
•	format_map()	Formats specified values in a string
•	index()	Searches the string for a specified value and returns the position of where it was found
•	isalnum()	Returns True if all characters in the string are alphanumeric
•	isalpha()	Returns True if all characters in the string are in the alphabet
•	isdecimal()	Returns True if all characters in the string are decimals
•	isdigit()	Returns True if all characters in the string are digits
•	isidentifier()	Returns True if the string is an identifier
•	islower()	Returns True if all characters in the string are lower case
•	isnumeric()	Returns True if all characters in the string are numeric
•	isprintable()	Returns True if all characters in the string are printable
•	isspace()	Returns True if all characters in the string are whitespaces
•	istitle()	Returns True if the string follows the rules of a title
•	isupper()	Returns True if all characters in the string are upper case
•	join()	Joins the elements of an iterable to the end of the string
•	ljust()	Returns a left justified version of the string

•	lower()	Converts a string into lower case
•	lstrip()	Returns a left trim version of the string
•	maketrans()	Returns a translation table to be used in translations
•	partition()	Returns a tuple where the string is parted into three parts
•	replace()	Returns a string where a specified value is replaced with a specified value
•	rfind()	Searches the string for a specified value and returns the last position of where it was found
•	rindex()	Searches the string for a specified value and returns the last position of where it was found
•	rjust()	Returns a right justified version of the string
•	rpartition()	Returns a tuple where the string is parted into three parts
•	rsplit()	Splits the string at the specified separator, and returns a list
•	rstrip()	Returns a right trim version of the string
•	split()	Splits the string at the specified separator, and returns a list
•	splitlines()	Splits the string at line breaks and returns a list
•	startswith()	Returns true if the string starts with the specified value
•	strip()	Returns a trimmed version of the string
•	swapcase()	Swaps cases, lower case becomes upper case and vice versa

•	title()	Converts the first character of each word to upper case
•	translate()	Returns a translated string using maketrans()
•	upper()	Converts a string into upper case
•	zfill()	Fills the string with a specified number of 0 values at the beginning

These functions are the real power of strings. Your algorithm or size of the program can be reduced by more than 50% if you use these inbuilt functions.

Using For loop with String

Since a for loop is a membership loop which initializes the loop variable by each member of the collection and String is also a collection of characters. So, this means that we can use for loop on strings too.

```
for i in 'python':
    print('letter -->',i)
```

OUTPUT

```
>>> letter --> p
    letter --> y
    letter --> t
    letter --> h
    letter --> o
    letter --> n
```

QUESTIONS ON STRING

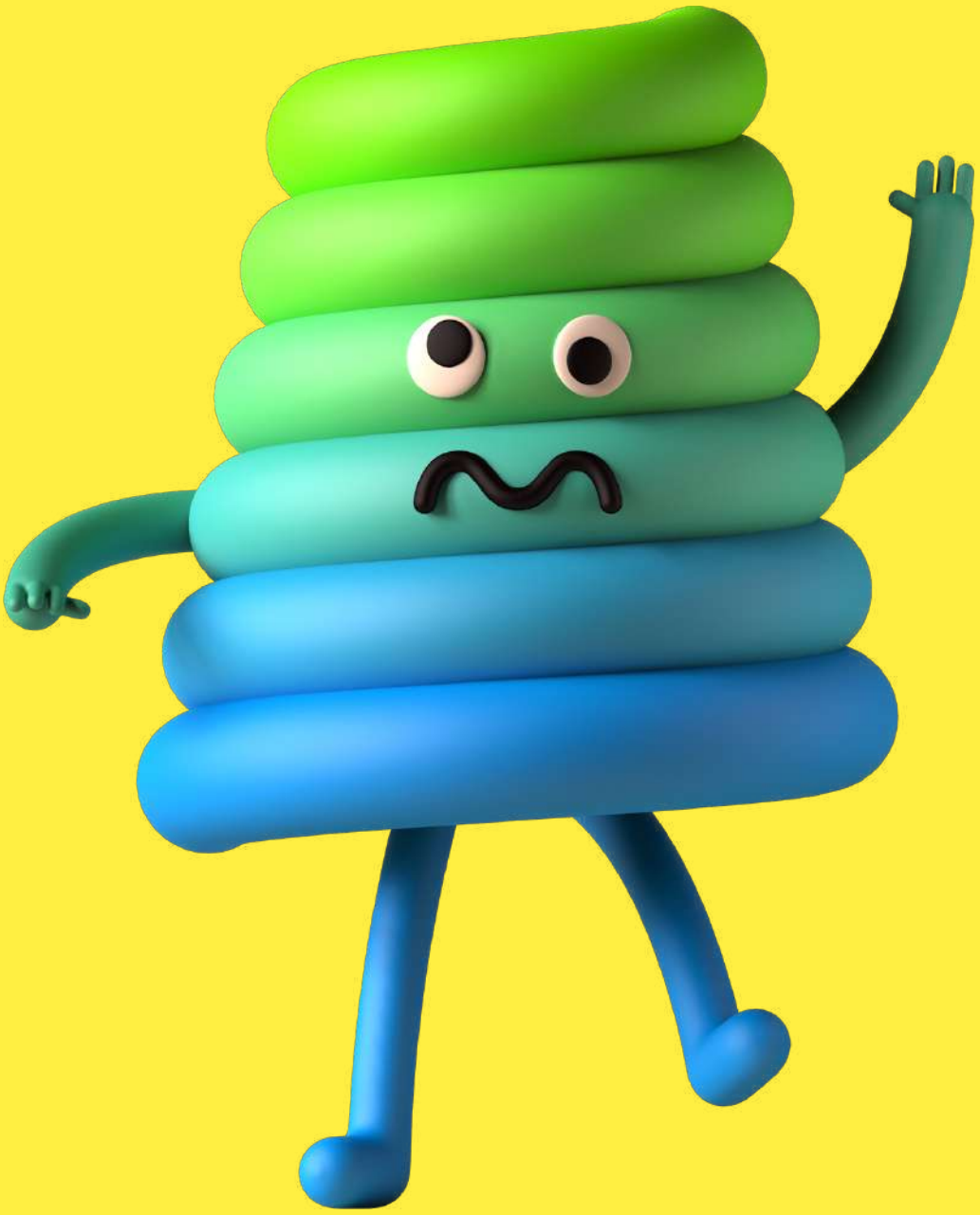
Q1) Run dir(str) and help(str) to get detail about all the methods of the strings.

Q2) Print the number of vowels in the string.

Q3) Check how many uppercase and lowercase characters are in the string.

CHAPTER 5

COLLECTIONS



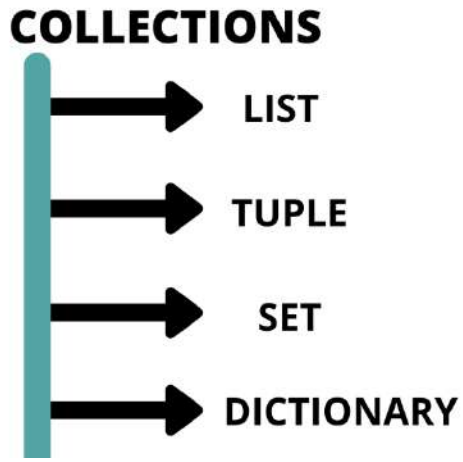
COLLECTIONS

Integers, float, complex, boolean all these are single strands of data.
Python inbuilt Collection data types is used to create a collection of one or more than one value.

These Collection data types are :

1) List
2) Tuple
3) Set
4) Dictionary

We will be seeing each and all of them one by one in this chapter.



LIST

A list is a heterogeneous collection of data, which means that it can have any type of data as a member of it.

It is created by square brackets. [ ]

It is mutable.

```
a = [1, 2, 3.9, 7+8j, 'value', True]
print(a)
```

OUTPUT

```
>>> [1, 2, 3.9, (7+8j), 'value', True]
```

String-Like INDEXING AND SLICING

```
a = [8,9,10]

print(a[0])
print(a[1])
print(a[2])
```

OUTPUT

```
>>> 8
    9
    10
```

```
print(a[-1])
print(a[-2])
print(a[-3])
```

OUTPUT

```
>>>  10
     9
     8
```

```
print(a[1:3])
```

OUTPUT

```
>>> [9, 10]
```

```
print(a[0:3:2])
```

OUTPUT

```
>>> [8, 10]
```

```
print(a[::3])
```

OUTPUT

```
>>> [8]
```

```
print(a[::-1]) # reversing
```

OUTPUT

```
>>> [10,9,8]
```

Lists are Mutable

The collections which allows the programmer to change or replace their internal values are known as mutable collections or values while the vice-versa is immutable.

IMMUTABLE

MUTABLE

List is mutable.

```
a = [8,9,10]
  #Original List
print(a[1])
```

OUTPUT

```
>>>  9
```

The value at index 1 → 9

```
a[1] = 9999
  #REPLACING VALUE AT POSITION 1 (9) by 9999.
print(a)
```

OUTPUT

```
>>> [8,9999, 10]
```

This shows the mutability feature of List.

Functions of List

	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

Taking **n** number of input in list

```
n = int(input('enter the number of elements you want '))
list1 = []

for i in range(0,n):
    value = int(input('enter the value '))
    list1.append(value)
```

_____OUTPUT_____

```
>>> enter the number of elements you want 3
    enter the value 12
    enter the value 31
```

```
print(list1)
```

_____OUTPUT_____

```
>>> [12,31,43]
```

Iterating through List using FOR

```
list1 = [1,'a',3,1,'b']
```

```
        for i in list1:
                print('value is',i)
```

OUTPUT

```
>>> value is 1
    value is a
    value is 3
    value is 1
    value is b
```

```
    #printing all even numbers in list inputted by user

    n = int(input('Enter the number of elements '))
    list2 = []
    for i in range(0,n):
        val = int(input('enter the value '))
        list2.append(val)


    print('Even numbers are--> ')
    for i in list2:
        if i%2==0:
            print(i)
```

OUTPUT

```
>>> Enter the number of elements 5
    enter the value 12
    enter the value 134
    enter the value 1213
    enter the value 1342
    enter the value 341
    Even numbers are-->
```

```
12
134
1342
```

TUPLE

A tuple is also a heterogeneous collection of data, which means that it can have any type of data as a member of it.

It is created by square brackets. ()
But the main difference from the list is that It is immutable.

```
tup = (1, 2, 3.9, 7+8j, 'value', True)
print(tup)
```

OUTPUT

```
>>> [1, 2, 3.9, (7+8j), 'value', True]
```

String-Like INDEXING AND SLICING

```
tup = ('a',3,9)

print(tup[0])
print(tup[1])
print(tup[2])
```

```
OUTPUT→ a
        3
        9
```

```
print(tup[-1])
print(tup[-2])
print(tup[-3])
```

```
OUTPUT→ 9
 3
 a
```

```
print(tup[1:3])
```

_____

_____OUTPUT_____

```
>>> (3,9)
```

```
print(tup[0:3:2])
```

_____

_____OUTPUT_____

```
>>> ('a',9)
```

```
print(a[::3])
```

_____

_____OUTPUT_____

```
>>> ('a')
```

```
print(a[::-1]) # reversing
```

OUTPUT

```
>>> (9,3,'a')
```

Tuples are Immutable

The biggest difference between tuple and list is that **lists are mutable while tuples are Immutable.**

```
tup = (4,5,6)
#Original Tuple
print(tup[1])
```

OUTPUT

```
>>> 5
```

The value at index 1 → 5

```
tup[1] = 9999
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-40-dc5d695c0277> in <module>
----> 1 tup[1] = 9999
```

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

We cannot change the value once created in the tuple.

This shows the immutability feature of Tuple.

Methods of Tuple

As we know it is immutable that means we will have very less functions as compared to list.

	Method	Description
•	count()	Returns the number of elements with the specified value
•	index()	Returns the index of the first element with the specified value

#Taking Input is not possible in Tuple

Iterating through Tuple using FOR

```
tup = (1,'a',3,1,'b')

for i in tup:
        print('value is',i)
```

OUTPUT

```
>>> value is 1
    value is a
    value is 3
    value is 1
    value is b
```

Set

A Set is an unordered and unique heterogeneous collection of different data types.

It is created by curly brackets. {}

```
l = list('pythonpython')
print(l)
```

_____OUTPUT_____

```
>>> ['p', 'y', 't', 'h', 'o', 'n', 'p', 'y', 't', 'h', 'o', 'n']
```

```
s = set('pythonpython')
print(s)
```

_____OUTPUT_____

```
>>> {'h', 'y', 'o', 'n', 't', 'p'}
```

X	NO INDEXING AND SLICING	X

Since, set is an unordered and unique collection so it doesn't have any fixed indexes which make it not possible to give an address to any value.

So, NEITHER INDEXING NOR SLICING ALLOWED.

IF WE TRY TO DO SO WE'LL GET TYPE ERROR.

```
s = set('pythonpython')
print(s[1])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-5ee4f18f93b6> in <module>
      1 s = set('pythonpython')
      2 print(s)
----> 3 print(s[1])

TypeError: 'set' object is not subscriptable
---------------------------------------------------------------------------
```

For Loop iterating Set

```
# for loop iterating SET

s = {'a','b','ae','gad','ad'}
for i in s:
    print('set value',i)
```

_____OUTPUT_____

```
>>> set value b
```

```
set value gad
set value ae
set value a
set value ad
```

Question. Find Frequency of each character in String using Set.

```
#frequency of all characters in string

word = input('Enter the word ')
s = set(word)

for i in s:
    print(i,'->',word.count(i))
```

OUTPUT

```
>>>Enter the word programmer
   m -> 2
   g -> 1
   o -> 1
   r -> 3
   a -> 1
   p -> 1
   e -> 1
```

In [ ]:

METHODS ON SET

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

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and other

•	union()	Return a set containing the union of sets
•	update()	Update the set with the union of this set and others

Sets are mutable

Set is a mutable collection, which means that you can add, remove or change particular value in it.

Original Set

```
s = {1,2,3}
s.add(4) # adding a value in set
print(s) # original set updated
```

OUTPUT

```
>>> {1, 2, 3, 4}
```

```
s.remove(1) # a value removed
print(s)    # original set removed
```

OUTPUT

```
>>> {2, 3, 4}
```

frozenset() method

As we mentioned above, that the sets are mutables. But there is a special way of converting mutable sets to immutable sets using the **frozenset()** method.

```
s = {1,2,3} #original set
fz = frozenset(s) # immutable set
print(fz)
```

OUTPUT

```
>>> frozenset({1, 2, 3})
```

only possible methods on frozenset are:

copy, difference, intersection, isdisjoint,
Issubset, issuperset, symmetric_difference, union from set..

DICTIONARY

A dictionary is a Key:Value pair collection. It is created by {} too.
It is basically a collection which is unordered, changeable and does not allow duplicates.

They are written with curly brackets, and have keys and values.

{ 'KEY' : 'VALUE' }

 # BASIC SYNTAX OF DICTIONARY

```
dictionary = { 'key' : 'value'}
```

#example of Dictionary

```
data = {
     'username':'A. Kumar',
     'designation': 'manager',
     'code':78612,
     'Salary':350000
        }
```

 # comparing list with dictionary

```
list1 = [8,9,10]
print(list1[1])
```

OUTPUT

```
>>>  9
```

```
# NAMED INDEXES

dict1 = {'a':8,'c':9, 'd':10}
print(dict1['c'])
```

_____

_____OUTPUT_____

```
>>> 9
```

Keys, Values and Items

{ KEY : VALUE , KEY:VALUE ..... }

A key in a dictionary is used to get the value in a dictionary.

The key:value pair are called Items.

A Key should always be an immutable value like string(str), tuple , numbers , etc.

A value can be literally any data type.

Accessing Values

A value is accessed by its key.

```
Example
```

```
    print the "color" value of the dictionary:

    vdict = {
        "type": "car",
        "color": "red",
        "year": 1964
            }
    print(vdict["color"])
```

_____OUTPUT_____

```
>>> red
```

It returns **KeyError** if the key is not present in the dictionary.

```
       vdict = {
               "type": "car",
               "color": "red",
               "year": 1964
               }

# printing non available key
     print(vdict['BRAND'])
```

```
>>> ---------------------------------------------------------------------------
---
KeyError                                  Traceback (most recent call last)
<ipython-input-23-27ea446097bf> in <module>
----> 1 print(vdict['BRAND'])

KeyError: 'BRAND'
```

Some Facts about Dictionary

- **Unordered**

  When we say that dictionaries are unordered, it means that the items does not have a defined order, you cannot refer to an item by using an index.
- **Mutable**

  Dictionaries are changeable, meaning that we can change, add or remove items after a dictionary has been created.
- **Duplicates Not Allowed**

  Dictionaries cannot have two items with the same key:

```
#EXAMPLE
data = {
  'username':'A. Kumar',
  'designation': 'manager',
  'salary':350000,
  'salary':123
      }
#Salary object is repeated twice but the latest one is added.
  print(data)
```

OUTPUT

```
>>> {'username': 'A. Kumar', 'designation': 'manager', 'salary': 123}
```

Methods on Dictionary

Dictionary has many important methods.

METHODS	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

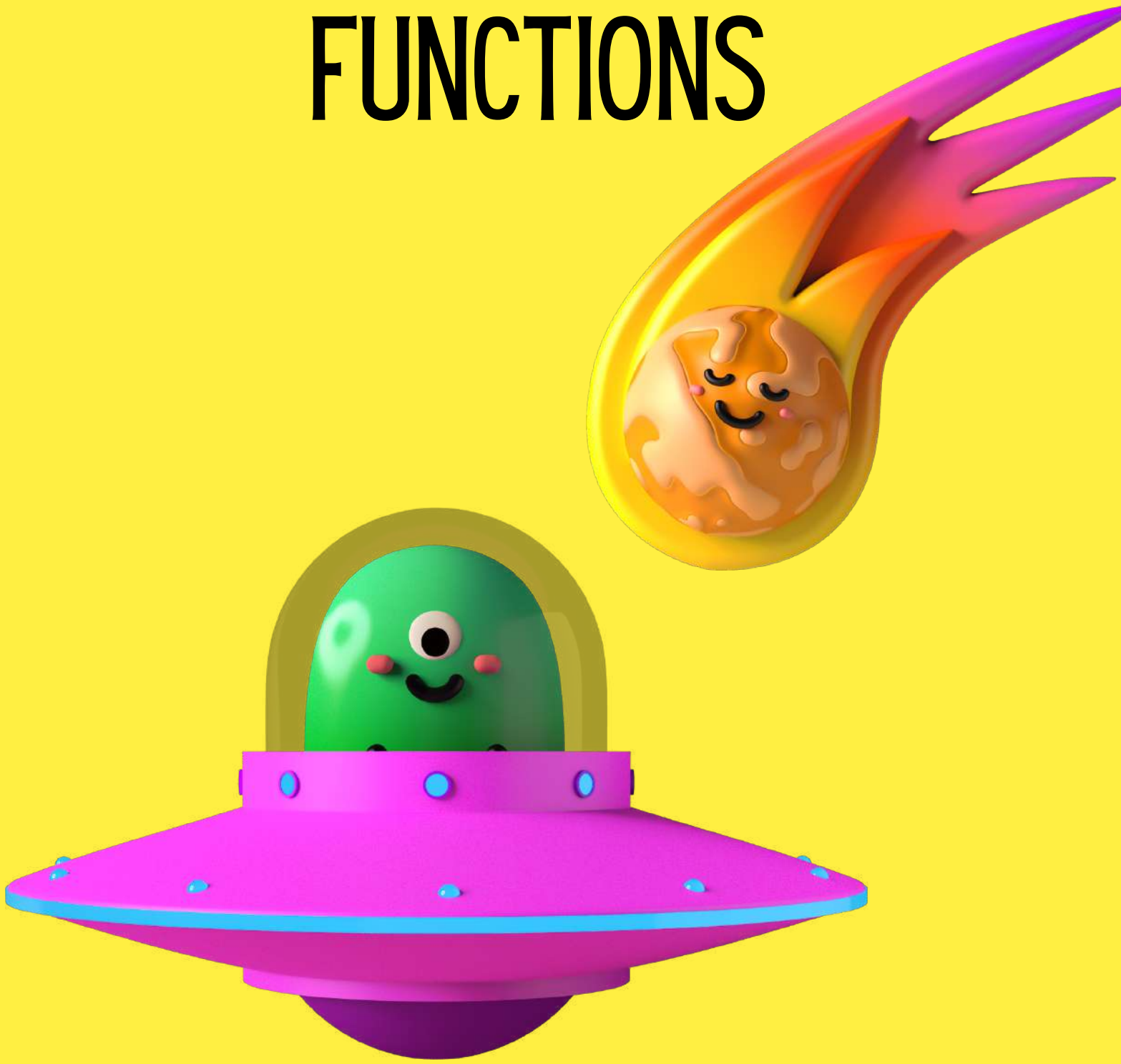
Q. CREATE AN USER LOGIN LOGIC WITH DICTIONARIES AS DATABASE.

```
database = {'user1':'pass1','user2':'pass2','user3':'pass3'}

username = input('enter the username ')

if username in database.keys():
   password = input('enter the password ')
   if database[username] == password:
       print('you are logged in')
   else:
       print('sorry wrong password')
else:
   print('you are not registered')
```

WELL DONE! YOU HAVE WORKED VERY HARD TILL NOW!!!
YOU DESERVE A FUNNY MEME.

CHAPTER 6

FUNCTIONS



FUNCTIONS

Dividing codes into smaller parts which performs a particular function, is known as functional programming. And the smaller parts are known as Functions.

Imagine instead of having a system of organs for every function, humans had a single organ for all requirements. Do you think that the human species would have survived for thousands of years ?
The Answer is NO.
Just Like we have functional systems for a particular task we can also create functions in our programs.

TO CREATE FUNCTIONS IN PYTHON ONE HAS TO USE **def** KEYWORD.

SYNTAX

```
def function_name(parameters):
    your code here
```

EXAMPLE

Let's create a function for adding two numbers which will return the value of summation.

```
def show():
    print('Hello World')
```

1000s lines of code
at a go.
depending on each other,

DIFFICULT TO ANALYSE BUGS

vs

a program with parts of code in it which can be acted as an indepented code too.

EASY TO ANALYSE BUGS

The Above created function will be saved in the memory of interpreter but will not work until and unless we don't call it by it's name.

CALLING A FUNCTION

Calling a function is very easy. We just have to write its name and give the appropriate values in parameters. This function above has print in it and no parameters. So simply write its name with brackets and it will run.

```
show()
```

_____

_____OUTPUT_____

```
>>> Hello World
```

Return Keyword

A return statement ends the execution of a function, and returns flow to the location where it was called with a value. Execution resumes in the calling function at the point immediately following the call. A return statement can return a value to the calling function.

```
def func():
    return 'value'
```

```
    a = func()
    print(a)
```

OUTPUT

```
>>> value
```

a variable was returned the value from function so it printed so.

```
def func():
   return 'value'
            |
        ____|
       |
 a = func()
 print(a)
```

If we try to store a non-return function in a variable we will get None in the Variable and the function will print the message because there is no return keyword.

If there is no return in the function we get none.

```
def show():
    print('hey')
value = show()
print(value)
```

OUTPUT

```
>>> hey
    None
```

ARGUMENTS / PARAMETERS

Information can be passed into functions as arguments.Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

```
def func():
    return 'Hello world'

a = func() # calling argument-less parameter
print(a)
```

OUTPUT

```
>>> Hello World
```

```
def func(word):
    return 'hello '+word

a = func('python') # calling function with parameter
print(a)
```

OUTPUT

```
>>> Hello python
```

DEFAULT PARAMETER

This is the concept of parameters in which we can provide the default value to them.

If the user doesn't input any value for the parameter that time the default value is used.

```
def func(word='world'): # value of word stays 'world'
    return 'hello '+word

a = func() # calling function with parameter
print(a)
```

_____OUTPUT_____

```
>>> Hello World
```

```
def func(word='world'): # value of word becomes 'python'
    return 'hello '+word

a = func('python') # calling function with parameter
print(a)
```

_____OUTPUT_____

```
>>> Hello python
```

Q. CREATE A LOGICAL SYSTEM USING FUNCTIONS AND DICTIONARIES FOR :
A) DISPLAY USERS
B) REMOVE USER
C) ADD USER
D) LOGIN AS A USER
E) CHANGE PASSWORD

```
database = {'user1':'pass1','user2':'pass2'}

def add():
    username = input('enter the username ')
    if username in database.keys():
        print('---------------------------')
        print('user already exist re-check')
        print('---------------------------')
    else:
        password = input('enter the password ')
        repasswd = input('confirm password ')
        if password == repasswd:
            database[username] = password
        else:
            print('passwords dont match try again')
    display()


def login():
    username = input('enter the username ')
    print('------------------------')
    if username in database.keys():
```

```
        password = input('enter the password ')
        if database[username]==password:
            print('LOGGED IN')
        else:
            print('Password is not Correct')
    else:
        print('404 User not found')
    print('-----------------------')


def change():
    username = input('enter the username ')
    print('-----------------------')
    if username in database.keys():
        password = input('enter the old password ')
        if database[username]==password:
            password = input('enter the password ')
            newpasswd = input('confirm password ')
            if password == newpasswd:
                database[username] = password
            else:
                print('passwords dont match try again')
        else:
            print('Password is not Correct')
    else:
        print('404 User not found')
    print('-----------------------')
    display()


def remove():
    username = input('enter the username ')
    print('-----------------------')
    if username in database.keys():
```

```
        password = input('enter password to remove user  ')
        if database[username]==password:
            database.pop(username)
            print('User Removed')
        else:
            print('Password is not Correct')
    else:
        print('404 User not found')
    print('-----------------------')
    display()



def display():
    print('-----------------')
    print('USERS')
    print('-----------------')
    for i in database.keys():
        print(i)
    print('-----------------')



print('---------------------------')
print('WELCOME TO DATABASE')
print('---------------------------')

while True:
    print('PRESS 1 TO DISPLAY USERS')
    print('PRESS 2 TO ADD USER IN THE SYSTEM')
    print('PRESS 3 TO LOGIN IN THE SYSTEM')
    print('PRESS 4 TO CHANGE PASSWORD')
    print('PRESS 5 TO REMOVE USER')
    choice = int(input('ENTER CHOICE '))
```

```
if choice in [1,2,3,4,5]:

    if choice == 1:
        display()
    elif choice == 2:
        add()
    elif choice == 3:
        login()
    elif choice == 4:
        change()
    else:
        remove()
else:
    print('-------------------')
    print('INVALID OPTION')
    print('-------------------')
k = input('PRESS Y TO CONTINUE ELSE ANY KEY TO EXIT\n')
print('-------------------')
if k=='Y' or k=='y':
    continue
else:
    print('--------------------')
    print('THANK YOU FOR USING')
    print('--------------------')
    break
```

```
---------------------------
WELCOME TO DATABASE
---------------------------
PRESS 1 TO DISPLAY USERS
PRESS 2 TO ADD USER IN THE SYSTEM
PRESS 3 TO LOGIN IN THE SYSTEM
PRESS 4 TO CHANGE PASSWORD
PRESS 5 TO REMOVE USER
ENTER CHOICE 1
-----------------
USERS
-----------------
user1
user2
-----------------
PRESS Y TO CONTINUE ELSE ANY KEY TO EXIT
N
------------------
--------------------
THANK YOU FOR USING
--------------------
```

CHAPTER 7

ERROR HANDLING

ERROR HANDLING

Errors

Before learning error handling, one should know the common types of errors.

Errors are basically divided into three major categories

1) Syntax Error → these are typing mistakes and have to be resolved by the user.

```
# syntax error
print 'message'
```

```
 File "<ipython-input-24-21182fbd6d61>", line 2
    print 'message'
                  ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean
print('message')?
```

```
------------------------------------------------------------------------------------------------------------
---

# indentation error

if True:
print('missed indentation')
```

```
 File "<ipython-input-25-427c6f2830c1>", line 4
    print('missed indentation')
        ^
IndentationError: expected an indented block
```

---

2) **Runtime Error** → these mistakes are correct in syntax but wrong in applications which can be handled by the interpreter.

```
#1. NameError

print(mango) # variable not defined
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-26-a8c3d2eb9690> in <module>
      1 #1. NameError
      2
----> 3 print(mango)
NameError: name 'mango' is not defined
```

```
#2.  ValueError

int('abcd') #gave characters in place of integer
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-27-4183f855a302> in <module>
      1 #2.  ValueError
      2
----> 3 int('abcd') #gave characters in place of integer

ValueError: invalid literal for int() with base 10: 'abcd'
```

#3. AttributeError

```
print('abc'.reverse()) #no method reverse in string
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-28-86cee781f2da> in <module>
      1 #3. AttributeError
      2
----> 3 print('abc'.reverse()) #no method reverse in string

AttributeError: 'str' object has no attribute 'reverse'
```

#4. TypeError

```
print(1 + '1') #integer type + str type
```

```
---------------------------------------------------------------------------
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-29-2f85fecc9626> in <module>
      1 #4. TypeError
      2
----> 3 print(1 + '1') #integer type + str type

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

#5. IndexError

```
s='python'
print(s[100])#no index 100
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-30-c1d06ebd8125> in <module>
      2
      3 s='python'
----> 4 print(s[100])#no index 100

IndexError: string index out of range
```

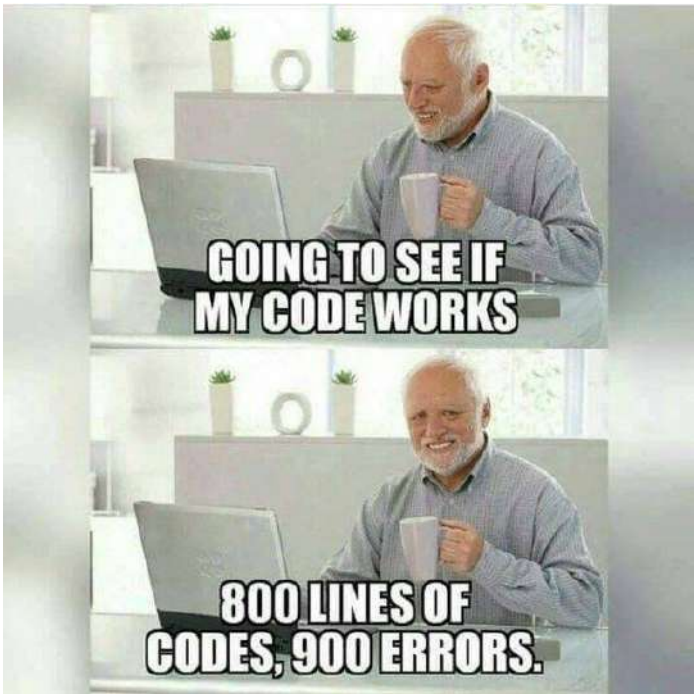
#6. KeyError

```
d  = {1:2,2:3}

print(d['value']) #no key like value
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-31-c25974faad1a> in <module>
      3 d  = {1:2,2:3}
      4
----> 5 print(d['value']) #no ket like value

KeyError: 'value'
```

---

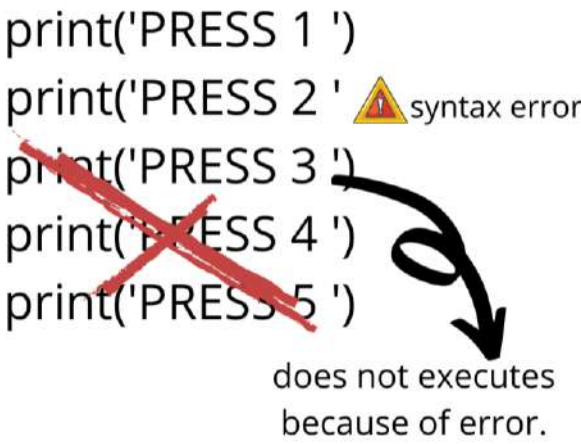
3) **Logical Error** → these mistakes are mathematical or logical in nature.



HANDLING ERROR

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the **try-except** statements



The **try** block lets you test a block of code for errors.

The **except** block lets you handle the error.

The **finally** block lets you execute code,

regardless of the result of the try- and except blocks.

```
#syntax of try-except block

    try:
        <code>
    except:
        <the code which runs if there is error in the code>
```

```
    # original code

      print('1'+1)   # adding string with int will give TypeError
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-08b4c50ea832> in <module>
      1 # original code
      2
----> 3 print('1'+1)  # adding string with int will give TypeError

TypeError: can only concatenate str (not "int") to str
---------------------------------------------------------------------------
```

```
      #handled code

      try:
          print('1'+1)
      except:
          print('Exception : you added a string in it')
```

_____OUTPUT_____

```
>>> Exception : you added a string in it
```

```
      # getting the type of code
```

```
try:
    print('python'.append([1,2,3]))
except Exception as e:
    print(e)
```

OUTPUT

```
>>> 'str' object has no attribute 'append'
```

```
# allowing certain types of error only

try:
    print('a' + 1)
except TypeError:
    print('type error in code')
```

OUTPUT

```
>>> type error in code
```

```
# allowing certain types of error only

try:
    print('a'.reverse())
except TypeError:
    print('type error in code')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-5-0060d911c001> in <module>
      2
```

```
      3 try:
----> 4     print('a'.reverse())
      5 except TypeError:
      6     print('type error in code')

AttributeError: 'str' object has no attribute 'reverse'
---------------------------------------------------------------------------
```

Q. USE TRY-EXCEPT IN LOGIN SYSTEM LOGIC.

```
database={'user1':'pass1','user2':'pas2','user3':'pass3'}

username = input('enter your username ')

try:
    db_pass = database[username]
    password = input('enter the password ')

    if db_pass == password:
        print('LOGGED IN')
    else:
        print('WRONG PASSCODE')
except KeyError:
    print('NO USER EXISTS LIKE THIS')
```

OUTPUT

```
enter your username user1
enter the password pass1
LOGGED IN

enter your username user1
enter the password pass3
WRONG PASSCODE

enter your username python
NO USER EXISTS LIKE THIS
```

CHAPTER 8

FILE HANDLING

FILE HANDLING

It is very difficult for a user to input a big amount of data through a terminal or shell of python.

But what is easy is the process of FILE HANDLING.

In which we can open txt,csv,doc,etc files in python and work accordingly on them.

We can either store(write) our data in them or retrieve(read) our data from them.

```
#SYNTAX OF OPENING A FILE

variable_name = open('*PATH OR FILE NAME*', '*Mode*')

# Modes of opening a file

'r' → read data

'w' → write and overwrite data

'a' → write and add data

'x' → creates a new file

'a+' → read and write from the file simultaneously
```

```
#Closing the file

variable_name .close()
```

Note: You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

'r' mode

we only read from a novel.
we dont write anything in it.

'w' mode

After every subject classs, we rub the board to overwrite new topics.

'a' mode

we keep appending or adding text in a notebook.
we dont delete previous

+

READING FILES

DEMO TEXT FILE

NAME OF FILE → data.txt

```
Hello everyone.
How are you all?
This is a fun way to learn python.
```

```
# reading a data from the file already existing

file = open('data.txt','r')
for i in file:
    print(i)
file.close()
```

_____OUTPUT_____

```
>>> Hello everyone.
    How are you all?
    This is a fun way to learn python.
```

You can work on the **i** variable all together treating it as a string and perform many functions on it.

Other methods of reading files include:-

```
#a read() method reads all the content of the file at once.


f = open("data.txt", "r")
print(f.read())
```

_____OUTPUT_____

```
>>> Hello everyone.
    How are you all?
    This is a fun way to learn python.
```

```
#You can read one line by using the readline() method.
#Read one line of the file:
```

```
        f = open("demofile.txt", "r")
        print(f.readline())
```

_____OUTPUT_____

```
>>> Hello everyone.
```

```
        #By calling readline() two times, you can read the two first
lines.
        #and for n lines run readline() n times.
        #Read two lines of the file:

        f = open("demofile.txt", "r")
        print(f.readline())
        print(f.readline())
```

_____OUTPUT_____

```
>>> Hello everyone.
    How are you all?
```

WRITING IN EXISTING FILES

To write in an existing file you must be writing the mode of file in open() as follows:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

.write() is used for writing in the files

DEMO TEXT FILE 2

NAME OF FILE → data.txt

Hello everyone.

APPENDING THE CONTENT IN OLD

```
#Example
#Open the file "data.txt" and add text to the file:

file = open('demofile2.txt', 'a')
file.write('More content added to file')
file.close()

#open and read the file after the adding:
file = open('data.txt', 'r')
print(file.read())
```

_____

_____OUTPUT_____

```
>>> Hello everyone.
    More content added to file
```

OVERWRITING THE OLD CONTENT

```
#Open the file "data.txt" and overwrite the content:

file = open('data.txt', 'w')
file.write('OLD DATA DELETED')
file.close()

#open and read the file after the overwriting:
file = open('data.txt', 'r')
print(f.read())
```

OUTPUT

```
>>> OLD DATA DELETED
```

#NOTE : 'w' DELETES THE PREVIOUS DATA AND OVERWRITES NEW

CREATING NEW FILES AND ADDING VALUES

We can create a new file in Python, using the open() method, with one of the following modes of file handling.

'x' - Create - will create a file, returns an error if the file exist

'a' - Append - will create a file if the specified file does not exist

'w' - Write - will create a file if the specified file does not exist

```
#Creating a file named 'data_new.txt'

file = open('data_new.txt', 'x')
#Result: a new empty file is created!


#Create a new file if it does not exist with 'w' and 'a'

file2 = open('data_new2.txt', 'w')
file3 = open('data_new3.txt', 'a')

#Result: 2 new empty files will be created
```

DEMO QUESTION

NAME OF FILE → database.txt

```
user1,pass1
user2,pass2
user3,pass3
user4,pass4
```

Que. We have a text file we have to get the data from the file in a dictionary as a key:value pair and create a login system logic.

SOLUTION :-

```
database = {}
file = open('database.txt','r')
for line in file:
    a = line.strip().split(',')
    database[a[0]] = a[1]
```

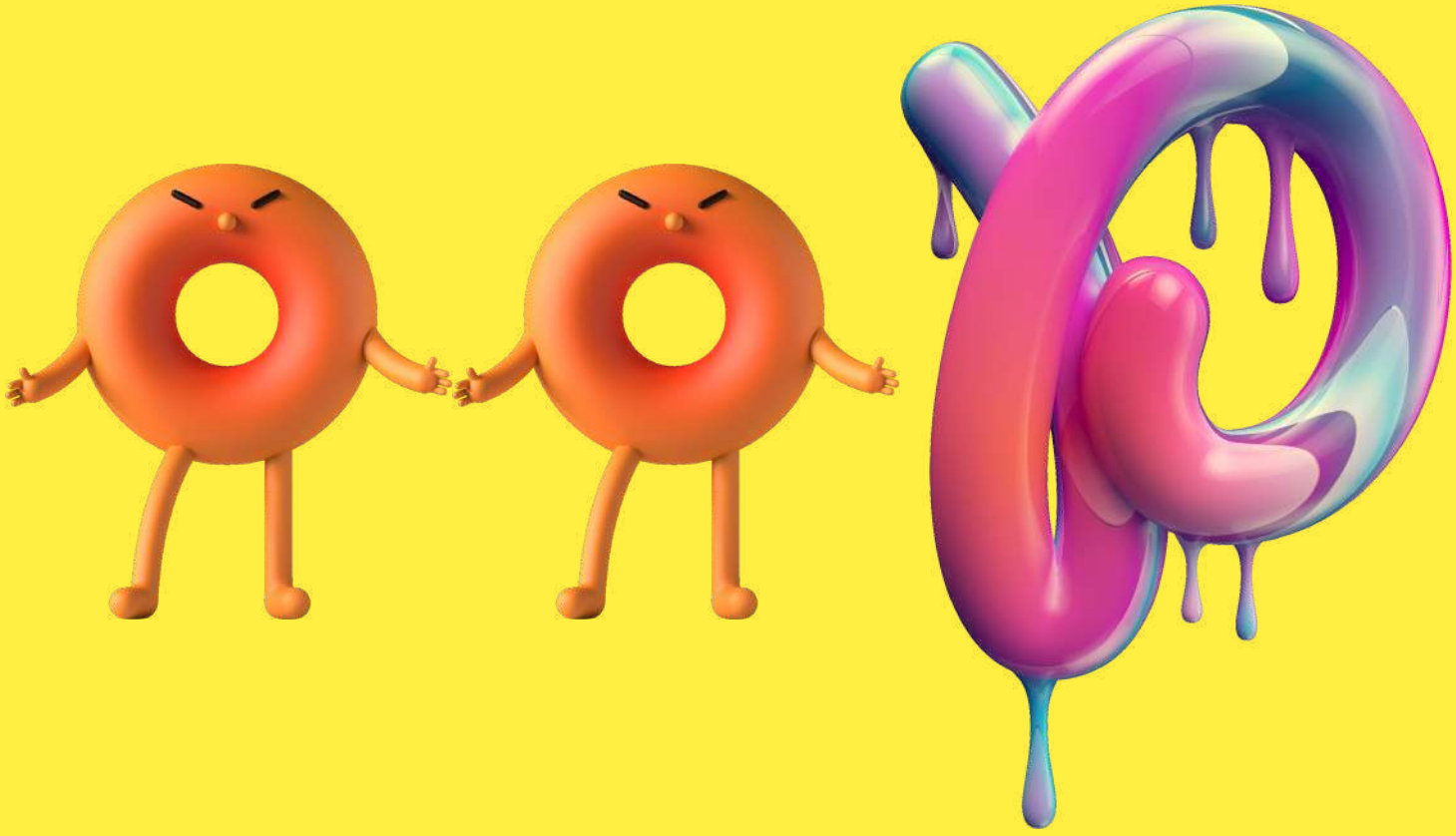
```
username= input('enter your username ')

try:
   db_pass = database[username]
   password = input('enter the password ')
   if password == db_pass:
       print('YOU ARE LOGGED IN')
   else:
       print('PASSWORD IS NOT CORRECT')
except KeyError:
   print('THE USER WAS NOT FOUND')
```

OUTPUT

```
>>> enter your username user1
    enter your password pass1
    YOU ARE LOGGED IN
```

CHAPTER 9

OOP



OBJECT IN PYTHON

OOP stands for **Object Oriented Programming** language, which is the process of implementing coding in terms of objects. Whereas an Object is a representation of a real life entity compromising state, behavior and properties.

Example of object → A car, because it has state(working), behavior(working, driving),

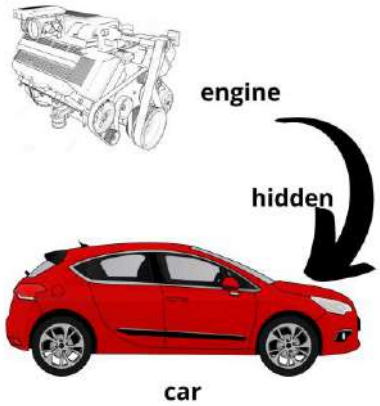
properties(color, model, etc.)

4 Principles of Object Oriented Programming

ABSTRACTION

The process of hiding the unnecessary details and showing the important ones only is known as abstraction.

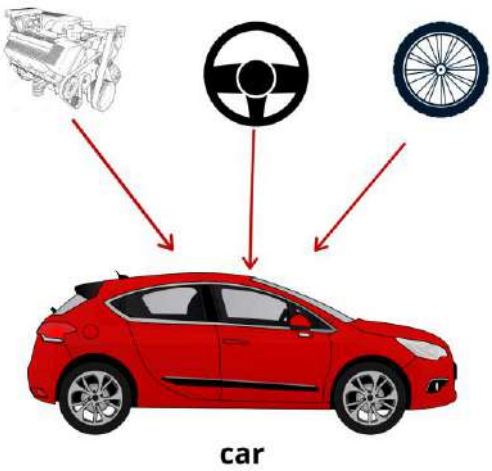
Eg. Hiding the engine of a car because a driver doesn't require to see it while driving.



ENCAPSULATION

The wrapping up of data into a single unit is known as encapsulation.

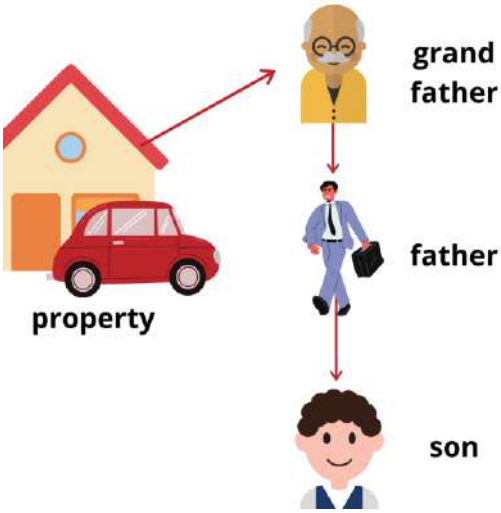
Eg. The wrapping up of multiple spare and important parts of a car into a single unit car.



INHERITANCE

The process of getting the properties from the pre-built source (parent) to a new source(child) is known as inheritance.

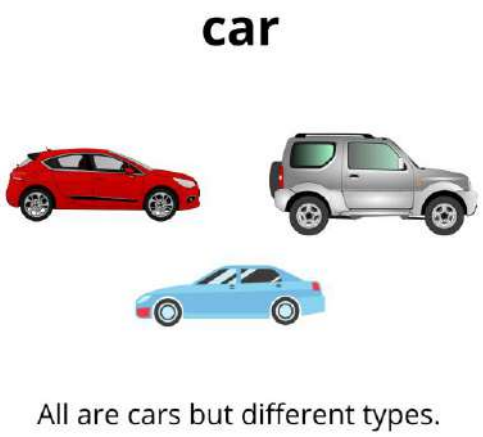
Eg. a new series car model might have inherited features from an old series car model.



POLYMORPHISM

The process of existing in multiple forms is known as polymorphism.

Eg. a car can exist in many forms, i.e., hatchback, sedan or suv.



All are cars but different types.

NOTE

1. Abstraction is achieved using encapsulation.
   When you'll wrap your data then only you'll be able to hide it.

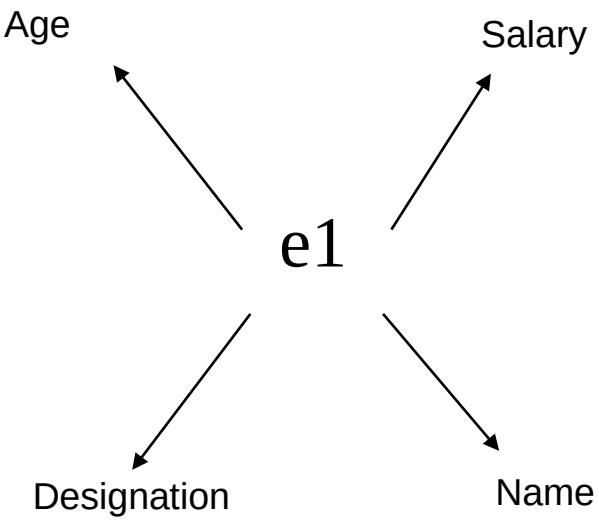
2. Everything in python is object (int,str, list and so on all are objects)

These principles are the reasons why we code in OOP basically.

Imagine having a list of 1000 employees with name, designation, salary and age in a list for all of them

emp_list = [['e1','manager',10000,21] ,....., ['e1000','worker', 9000, 34]]

The displayed list is not much easy to use, but the OOPs will help us to create an object for each element.



Lets say e1 is an object it will have name, age, designation and salary as properties.

Now we can get data easily by just mentioning :

name as e1.name

Age as e1.age

So, object.property will give me the values.

How easy it is to now list of variables which have objects like [e1,e2,e3,…….,e4].

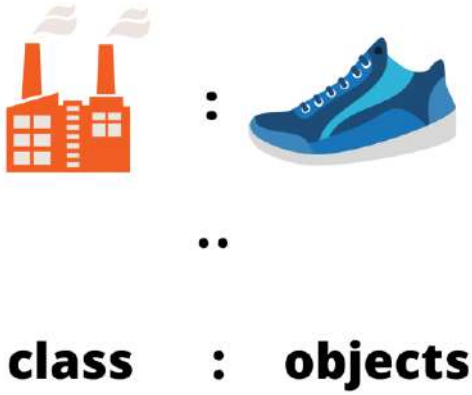
e1 as an object hid the unnecessary information (abstraction), wrapped all data into a single unit(encapsulation), can be used in further cases(inheritance) and can exist in many forms like many employees(polymorphism).

Creating an object → Concept of classes

A class is an object factory. It is like a blueprint to the structure or constituent of every object.

And an object is created by its class name.

In comparison, we can say that what is a shoe-factory to a shoe is what a class is to an object,



class → it's a keyword used to create a class.

```
class name_of_class:
    #code
    #property_builder function or constructor
    #behaviour builder function or methods

# an object will be created by this class name

object1 = name_of_class()
```

The above is the syntax of a normal class and object creation procedure.

```
class abc:
a = 5
#Here, our class name is abc. And it has a variable as property a.
obj1 = abc()
#Created object obj1 by class's name.
print(obj1.a)
```

_____OUTPUT_____

```
>>> 5
```

The object shows the properties of class here. Thus, python can be fully OOP too.

Current Calling Object and Constructor

As we know that an object is created by class so, for the same we can build our class with a property builder or we can call it constructor using the keyword **__init__()**. This keyword takes the parameter as the current calling object (means which object has called the class and wants the properties) and gives it the properties according to the blueprint. By convention we call the current calling object as **self**.

```
class employee:

    def __init__(self):
        self.name = 'Mahesh'
        self.age = 19

    def display(self):
        print(self.name,self.age)
```

A class with the property builder(constructor __init()) and behaviour (display()).
self is a current calling variable. When emp1 called the class self was emp1 when emp2 called the class self was emp2.

Example

```
emp1 = employee() # creating object
print(emp1.name)  # accessing properties
```

_____OUTPUT_____

```
>>> Mahesh
```

```
emp1.display()    # accessing behaviour
```

_____OUTPUT_____

```
>>> Mahesh 19
```

Giving user input values through __init__()

__init__() is accessed first as soon as we call the class. It is nothing but a function so we can recall how we used to use the concept of parameters in function and implement it over here in __init__().

```
class employee:

    def __init__(self, cname, cage):
        self.name = cname
        self.age = cage

    def display(self):
        print(self.name,self.age)
```

The above class has a constructor __init__() which has values sent from the user.

```
n = input('enter name--->  ') # creating values
a = input('enter age --->  ')
```

_____OUTPUT_____

```
>>> enter name--->  Kumar
    enter age --->  34
```

```
emp1 = employee(n,a) # object creation
print(emp1.name)
```

_____OUTPUT_____

```
>>> Kumar
```

```
emp1.display()
```

_____OUTPUT_____

```
>>> Kumar 34
```

Q. CREATE A CLASS OF BIKE AND STORE THE OBJECTS VALUE OF 3 BIKES IN A LIST.

```
class bike:

    def __init__(self, cmod, col):
        self.model = cmod
        self.color = col

    def bill(self):
        print('--------------------')
        print('model :',self.model)
        print('color :',self.color)
        print('--------------------')
```

THE ABOVE CODE IS CLASS CREATED.

```
list_of_bikes = []  # an empty list

for i in range(0,3):
    m = input('enter bike model ')
    c = input('enter bike color ')
    obj = bike(m,c)  # creating a new object
    list_of_bikes.append(obj)  # appending that into list everytime
```

OUTPUT→ enter bike model Yamaha r15
enter bike color blue
enter bike model bullet 350
enter bike color black
enter bike model ninja
enter bike color green

We will now see the list outcome below

```
print(list_of_bikes)
```

OUTPUT →

[<__main__.bike object at 0x0000016E70074488>,
<__main__.bike object at 0x0000016E70074848>,
<__main__.bike object at 0x0000016E7007D9C8>]

NOW WE WILL USE THESE OBJECTS INSIDE LIST TO ACCESS THE BEHAVIOUR bill()

```
for i in list_of_bikes:
    i.bill()
```

----------OUTPUT----------

```
--------------------
model : Yamaha r15
color : blue
--------------------
--------------------
model : bullet 350
color : black
--------------------
--------------------
model : ninja
color : green
--------------------
```

THIS WAS A SIMPLE EXAMPLE ON HOW WE IMPLEMENT OOPS IN NORMAL PROGRAMS.

OOP is an important skill in Django and Development Careers.

CHAPTER 10

TRICKS AND TIPS

TRICKS AND TIPS

Python programming is an important skill in this decade. And if you have reached here completing all the above concepts, then I am pretty much sure that you are thoroughly aware of the concepts of python as well as coding.

Now as a python programmer, I am sharing few tips for being better in python and also some syntaxial advanced concept of python which will surely help you to **fill the difference in knowing how to code and knowing what to code.**

FLATTEN IF ELSE (TERNARY STATEMENT)

A flatten if else is basically short-hand typing of real if-else condition for small conditions.

```
<when condition true> if <condition> else <when condition false>
```

The syntax is as above. It is basically a statement which first checks the condition, and the output depends on the same. If the condition is true the LHS side value (left to IF) will run, otherwise RHS side value (Right to else) will run.

```
n = int(input('enter a number--> '))
print('negative' if n<0 else 'positive')
```

```
OUTPUT→ enter a number--> -1
        negative
```

```
n = int(input('enter a number--> '))
print('negative' if n<0 else 'positive')
```

```
OUTPUT→ enter a number--> 600
        positive
```

LIST COMPREHENSION

We can use for loop inside the lists to create or generate values.

```
a = [value for value in collection]
```

```
a1 = [ i for i in range(1,6)]
print(a1)
```

```
OUTPUT→ [1,2,3,4,5]
```

```
a2 = [i*i for i in range(1,6)]
print(a2)
```

```
OUTPUT→ [1,4,9,16,25]
```

Lambda Function

Anonymous or one line function is created with the help of the lambda keyword.

```
f = lambda value : operation on value
```

```
F will be the variable which will work like the function.
```

```
f = lambda x: x**2
print(f(5))
print(f(10))
```

```
OUTPUT→ 25
        100
```

```
f2 = lambda x,y : x*y
print(f2(3,4))
print(f2(100,9))
```

```
OUTPUT→ 12
        900
```

TIPS FOR GETTING BETTER IN PYTHON.

1. **Writing codes in a notebook initially to understand indentation. Use a ruler and a pencil to create lines for the same.**
   When I shifted from Java to Python, the biggest obstacle for me was the new concept of indentation. So, to master it and understand it thoroughly I started using a notebook to write my code then used to check them on the computer.

2. **Create a timetable**
   Decide when you will code. Coding without a proper schedule might not result good for beginners. Give at least 30-60 minutes a day for practicing.

3. **NEVER SKIP A CODE DAY!**
   Consistency is the key, to master anything.
   Do small but at least do.

4. **Participate in Contests.**
   Create accounts on websites like HackerRank, Leetcode, Codechef and compete with others.

5. **Use Visualizer for your code.**
   Visit Pythontutor.com and run your code to debug it.

6. **Work on Projects**
   It doesn't need to be a big or huge collection of codes. It should be something with which you can implement your learning.

7. **Practice, Practice and more Practice**
   We don't need to learn our mother tongue. We start speaking it because it is what we are practicing. So, the only path to get to full knowledge of a concept is practicing that.

‘What you think, you become. What you feel, you attract. What you imagine, you create.’

–Buddha

THANK YOU FOR CHOOSING THIS BOOK.

More About the author.

"Priyam Kapoor, 19, has already taught and helped 500+ people in learning coding and technologies in his own way of relating things with reality.
Being a student and instructor at the same time gives the ability to think things in more than one way.

With the release of this ebook, they are already working on their upcoming projects.

You can connect with him on Linkedin and also on other social platforms.

Priyam Kapoor
priyam.kapoor2119@gmail.com