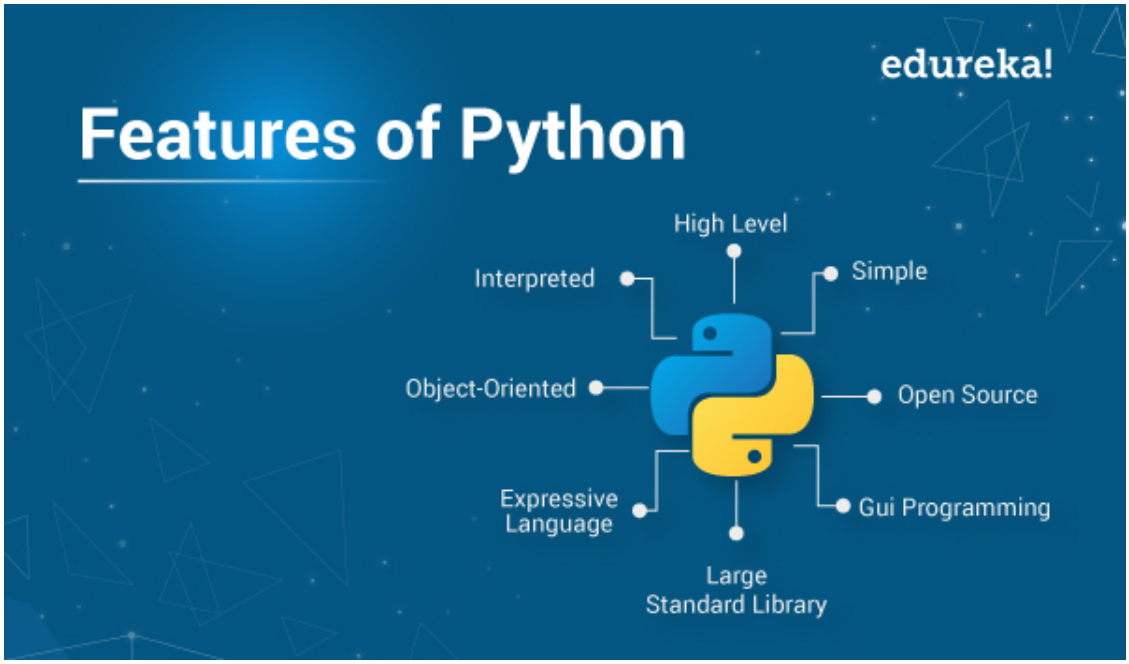
Python

Python is a general-purpose, interpreted, interactive, object- oriented and high-level programming language. It was created by Guido van Rossum, and released in 1991.

Python Features / Why to learn Python:



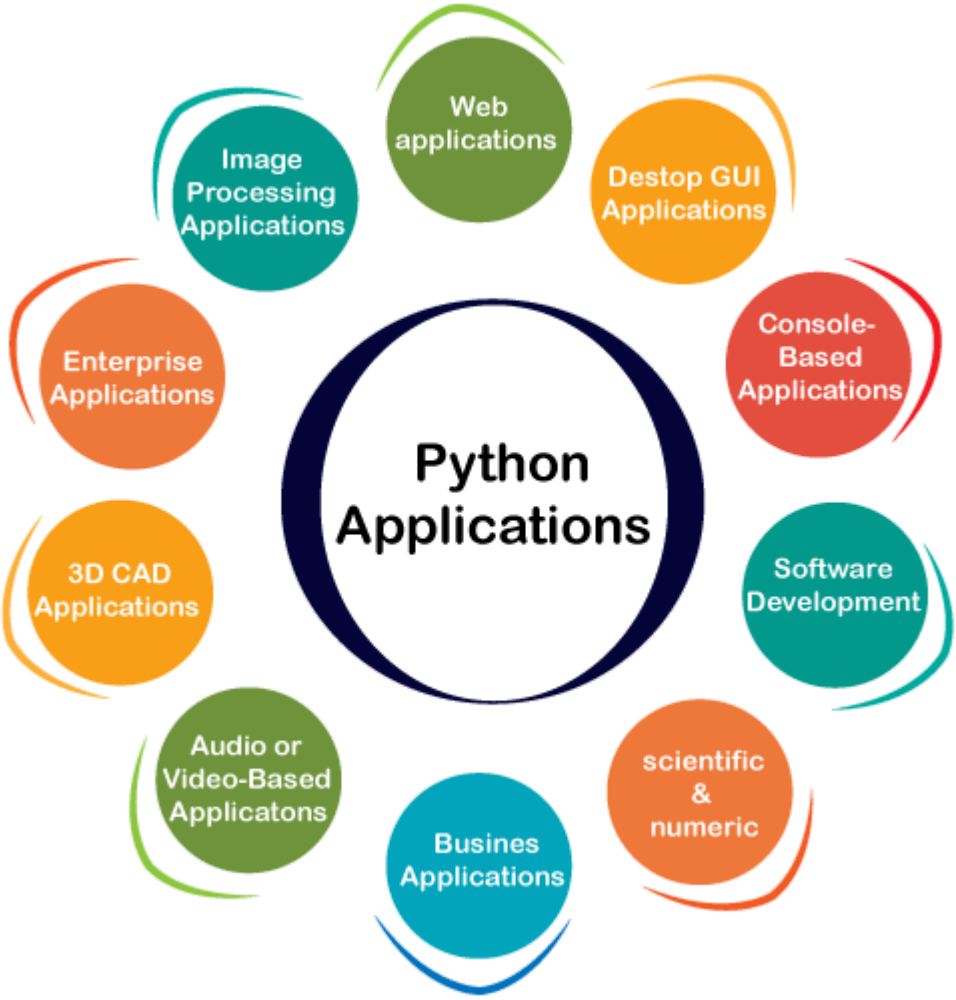
1. **Simple / General-purpose:**
   Python is usually developed for researchers. So, it is easy to learn by some researcher. So researcher does not feel difficulty to learn a programming language. The syntax is similar to the English language. Python syntax that allows developer to write program with fewer lines than some other programming language.
2. **Free and Open Source:**
   Python language is freely available at the official website and you can download it in free or without pay any single penny.
   Since it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.
3. **GUI Programming Support:**

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python.
PyQt5 is the most popular option for creating graphical apps with Python.

4. **Large Standard Library:**
   Python has a large standard library which provides a rich set of module and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers, etc.
5. **Expressive Language:**
   Python is an expressive language. Expressive in this context means that a single line of Python code can do more than a single line of code in most other languages. The advantages of a more expressive language are obvious – the fewer lines of code you write, the faster you can complete the project. Not only that – the fewer lines of code there are, the easier the program will be to maintain and debug.
6. **Object-oriented language:**
   Python supports object oriented style or technique of program that encapsulate code within objects. Python supports object-oriented language and concepts of classes, objects encapsulation, etc.
7. **Interpreted language:**
   Python is processed at run time by the interpreter you do not need to compile your program before executing it
   NOTE: Interpreter are used to convert a program written in a high level language into machine code that understood by computers

==========

Application of Python:



Python is used in many application domains. Here's a sampling

1. **Web Development:**
   Python can be used to make web-applications at a rapid rate. Why is that? It is because of the frameworks Python uses to create these applications. There is common-backend logic that goes into making these frameworks and a number of libraries that can help integrate protocols such as HTTPS, FTP, SSL etc. and even help in the processing of JSON, XML, E-Mail and so much more. Some of the most well-known frameworks are Django, Flask, Pyramid.
2. **Desktop GUI:**
   We use Python to program desktop applications. It provides the Tkinter library that can be used to develop user interfaces. There are some other useful toolkits such as the wxWidgets, Kivy, PYQT that can be used to create applications on several platforms. You can start out with creating simple applications such as Calculators, To-Do apps and go ahead and create much more complicated applications.
3. **Business Applications:**
   Business Applications are different than our normal applications covering domains such as e-commerce, ERP and many more. They require applications which are scalable, extensible and easily readable and Python provides us with all these features. Platforms such as Tryton is available to develop such business applications.
4. **Audio and Video Applications:**
   We use Python to develop applications that can multi-task and also output media. Video and audio applications such as TimPlayer, Cplay have been developed using

Python libraries. They provide better stability and performance in comparison to other media players.

5. **3D CAD Applications:**
   Computer-Aided Designing is quite challenging to make as many things have to be taken care of. Objects and their representation, functions are just the tip of the iceberg when it comes to something like this. Python makes this simple too and the most well-known application for CAD is Fandango.
6. **Game Development:**
   Python is also used in the development of interactive games. There are libraries such as PySoy which is a 3D game engine supporting Python 3, PyGame which provides functionality and a library for game development. Games such as Civilization-IV, Disney's Toontown Online, Vega Strike etc. have been built using Python.
7. **Machine Learning and Artificial Intelligence:**
   Machine Learning and Artificial Intelligence are the talks of the town as they yield the most promising careers for the future. We make the computer learn based on past experiences through the data stored or better yet, create algorithms which makes the computer learn by itself. The programming language that mostly everyone chooses? It's Python. Why? Support for these domains with the libraries that exist already such as Pandas, Scikit-Learn, NumPy and so many more.

==========

Python Installation

Follow these steps to download the full installer:

Step 1:

1. Open a browser window and navigate to the Python.org Downloads page for Windows.
2. Under the "Python Releases for Windows" heading, click the link for the Latest Python 3 Release - Python 3.x.x. As of this writing, the latest version was Python 3.8.4.
3. Scroll to the bottom and select either Windows x86-64 executable installer for 64-bit or Windows x86 executable installer for 32-bit.

   When the installer is finished downloading, move on to the next step.

Step 2: Once you've chosen and downloaded an installer, run it by double-clicking on the downloaded file. A dialog box like the one below will appear:

There are four things to notice about this dialog box:

1. The default install path is in the AppData/ directory of the current Windows user.
2. The Customize installation button can be used to customize the installation location and which additional features get installed, including pip and IDLE.
3. The Install launcher for all users (recommended) checkbox is checked default. This means every user on the machine will have access to the py.exe launcher. You can uncheck this box to restrict Python to the current Windows user.
4. The Add Python 3.8 to PATH checkbox is unchecked by default. There are several reasons that you might not want Python on PATH, so make sure you understand the implications before you check this box.

==========

Working in Python

Once you have Python installed on your system, you are ready to work on it. You can work in Python in two different modes:-

1. **Interactive Mode:** In this mode, you type one command at a time and Python executes the same. Python's interactive interpreter is also called Python Shell.
2. **Script Mode:** In this mode, we save all our commands in the form of a program file and later run the entire script. After running the script, the whole program gets compiled and you'll see the overall output.

==========

First Python program

As we are just getting started with Python, we will start with the most fundamental program which would involve printing a standard output to the console. The print() function

is a way to print to the standard output. The syntax to use print() function is as follows:-

Syntax:

In [ ]:
```
print(<Objects>)
```

- **Objects:** means that it can be one or more comma-separated 'Objects' to be printed.
- **Objects:** must be enclosed within parentheses.

The **print()** function prints the specified message to the screen, or other standard output device.

NOTE: print() function automatically brings you to next line.

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

```
Hello Python World!
```

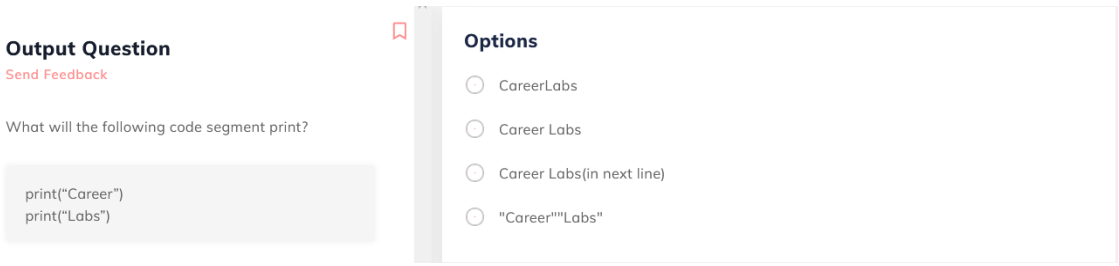
Python executed the first line by calling the print() function. The string value of Hello, Python World! was passed to the function

In [2]:
```
print("Hello")
print("Python World!")
```

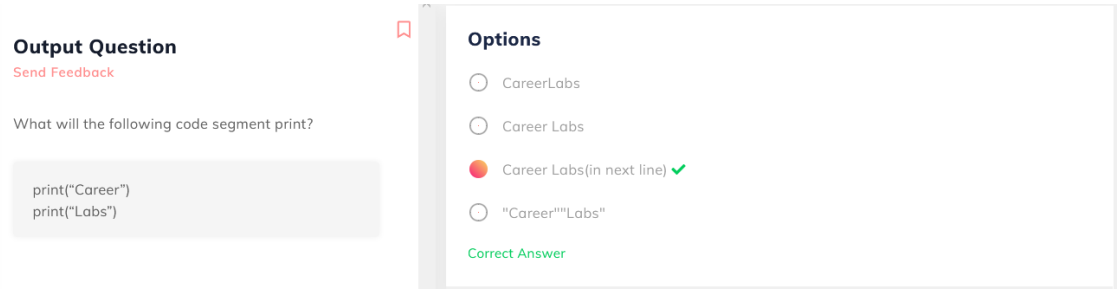
```
Hello
Python World!
```

When you run this code, the ending .py(If you stored in a .py file) indicates that the file is a Python program. Your editor then runs the file through the Python interpreter, which reads through the program and determines what each word in the program means. For example, when the interpreter sees the word print, it prints to the screen whatever is inside the parentheses. As you write your programs, your editor highlights different parts of your program in different ways. For example, it recognizes that print is the name of a function and displays that word in green. It recognizes that “Hello Python world!” is not Python code and displays that phrase in red. This feature is called syntax highlighting and is quite useful as you start to write your own programs.

Quiz



Solution



Explore via coding:

Write a python program to print your name,weight and height every attribute should come in new line

In [3]:
```
print("My name is sachin kapoor")
print("Weight is 56Kg")
print("Height is 168 CM")
```

```
My name is sachin kapoor
Weight is 56Kg
Height is 168 CM
```

==========

Variable

What are variables:

A variable in Python represents a named location that refers to value and whose values can be used and processed during the program run. In other words, variables are labels/names to which we can assign value and use them as a reference to that value throughout the code.

Variables are fundamental to programming for two reasons:

- Variables keep values accessible: For example, The result of a time-consuming operation can be assigned to a variable, so that the operation need not be performed each time we need the result.
- Variables give values context: For example, The number 56 could mean lots of different things, such as the number of students in a class, or the average weight of all students in the class. Assigning the number 56 to a variable with a name like num_students would make more sense, to distinguish it from another variable average_weight, which would refer to the average weight of the students. This way we can have different variables pointing to different values.

How are Values Assigned to A Variable:

Values are assigned to a variable using a special symbol “=”, called the assignment operator. An operator is a symbol, like = or +, that performs some operation on one or more values. For example, the + operator takes two numbers, one to the left of the operator and one to the right, and adds them together. Likewise, the “=” operator takes a

value to the right of the operator and assigns it to the name/label/variable on the left of the operator.

In python variables are created when you assign a value to it.
Variables are container for storing data values unlike other programming language, Python has no command for declaring a variable.

Example:

In [4]:
```
message = "Hello Python World!"  #assign "Hello Python world!" string to
print(message)
```

```
Hello Python World!
```

Variables do not need to be declared with any particular type and can even change type after they have been set.

Example:

In [5]:
```
message=4                          #assign message variable with 4
message="Hello Python World!"      #change the assignment of variable fr
print(message)
```

```
Hello Python World!
```

Rules for variable Naming convention:

When you're using variables in Python, you need to adhere to a few rules and guidelines. Breaking some of these rules will cause errors; other guidelines just help you write code that's easier to read and understand. Be sure to keep the following variable rules in mind:

1. Variable names can contain only letters, numbers, and underscores. They can start with a letter or an underscore, but not with a number. For instance, you can call a variable message_1 but not 1_message.
2. Spaces are not allowed in variable names, but underscores can be used to separate words in variable names. For example, greeting_message works, but greeting message will cause errors.
3. Avoid using Python keywords and function names as variable names; that is, do not use words that Python has reserved for a particular programmatic purpose, such as the word print.
4. Variable names should be short but descriptive. For example, name is better than n, student_name is better than s_n, and name_length is better than length_of_persons_name.

Example:

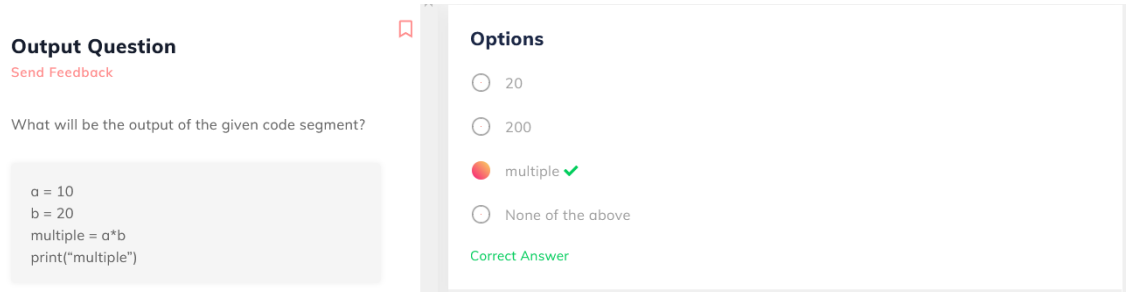
In [6]:
```
#Legal variable names:
myvar="sachin"
my_var="sachin"
_my_var="sachin"
MYVAR="sachin"
myvar1="sachin"

#Illegal variable names:
```

```
#all this variable will return you error
2myvar="sachin"
my-var="sachin"
my var="sachin"
```

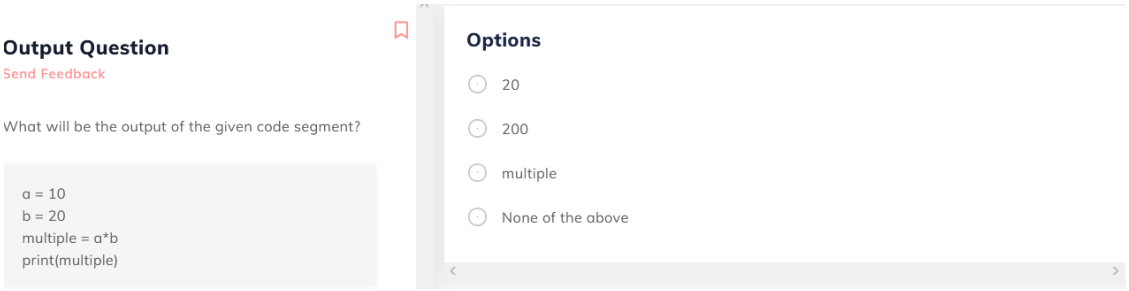
```
  File "<ipython-input-6-678872c366ec>", line 10
    2myvar="sachin"
     ^
SyntaxError: invalid syntax
```

Quiz

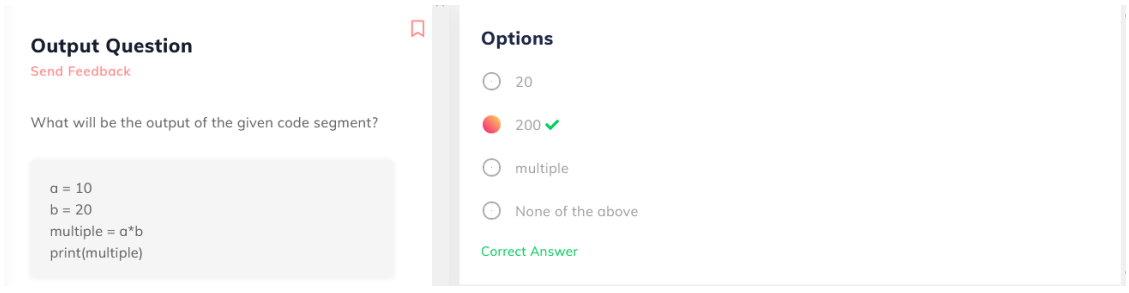
Solution



Quiz

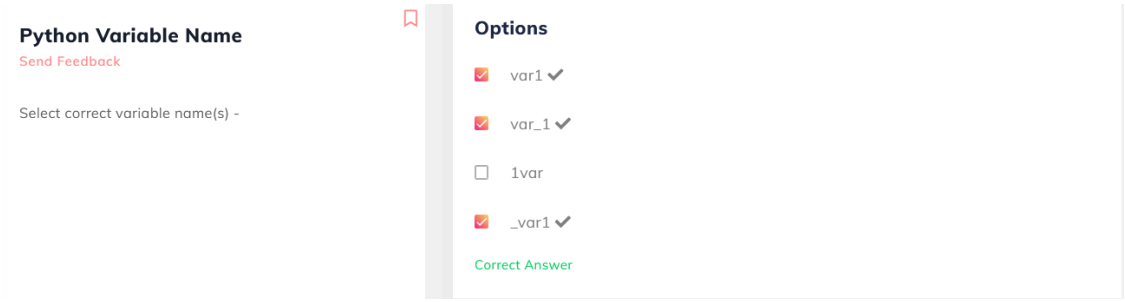


Solution

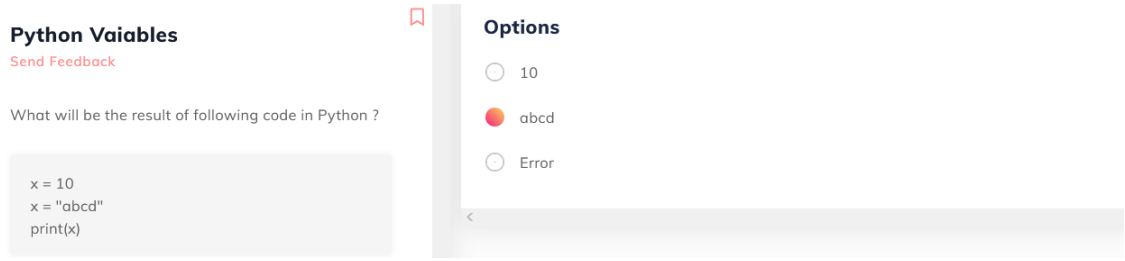


Quiz

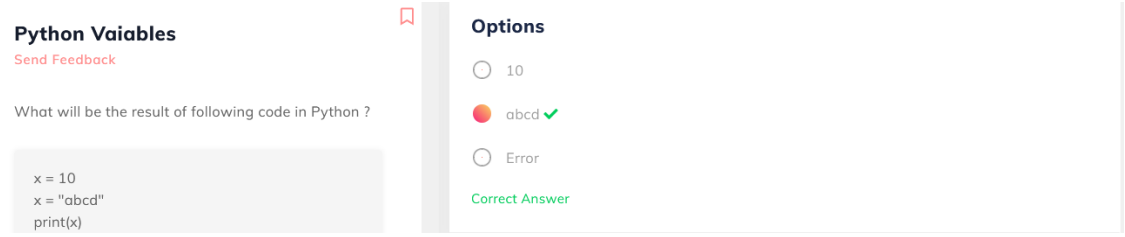
Solution



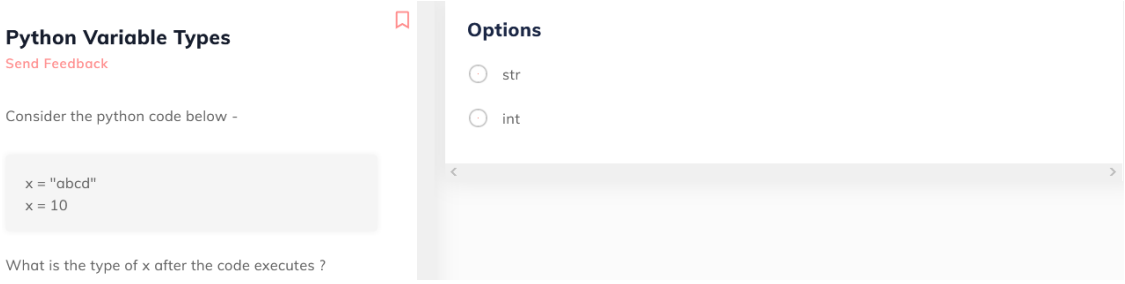
Quiz



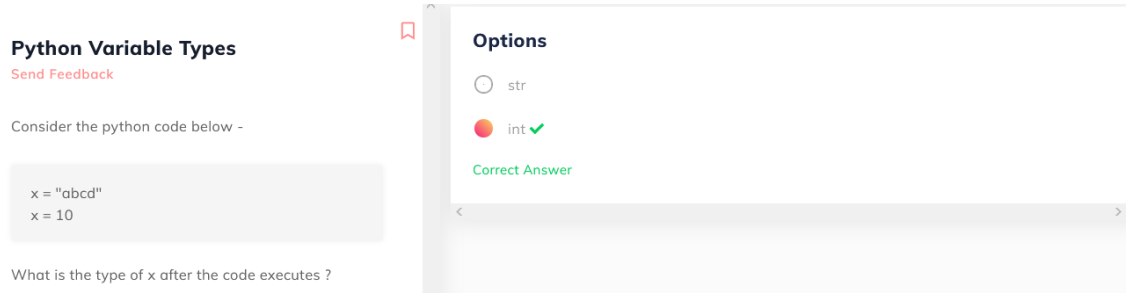
Solution



Quiz



Solution



Explore via coding:

Write a python program to store name,weight and height attribute in variable and print in on screen.

In [7]:
```
name="Sachin Kapoor"
weight=56
height=167

print("My name is: ",name)
print("Weight is: ",weight)
print("Height is: ",height)
```

```
My name is:  Sachin Kapoor
Weight is:  56
Height is:  167
```

Assign value to multiple variables:

Python allows you to assign multiple variables in one line.

Example:

In [8]:
```
X,Y,Z="orange","Banana","Cherry"
print(X)
print(Y)
print(Z)
```

```
orange
Banana
Cherry
```

and you can assign same value to multiple variables in one line.

Example:

In [9]:
```
X=Y=Z="orange"
print(X)
print(Y)
print(Z)
```

```
orange
orange
orange
```

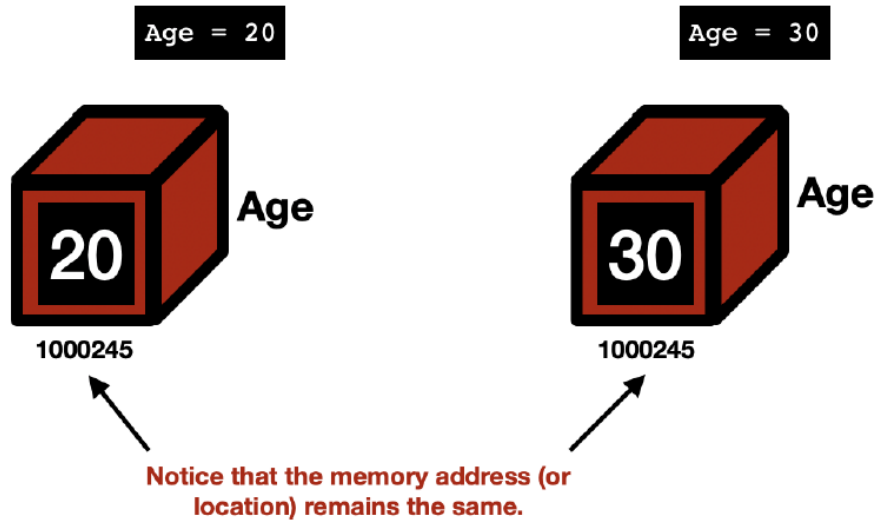
==========

Traditional Programming Languages’

Variables in Memory

Let us study how variables and the values they are assigned, are represented in memory, in traditional programming languages like C, C++, Java, etc. In these languages, variables are like **storage containers**. They are like named storage locations that store some value. In such cases, whenever we declare a new variable, a new storage location is given to that name/label and the value is stored at that named location. Now, whenever a new value is reassigned to that variable, the storage location remains the same. However, the value stored in the storage location is updated. This can be shown from the following illustration.

In [10]:
```
#Consider the following script:

Age = 20
Age = 30 # Re-assigning a different value to the same variable
```



In the above script, when we declare a new variable Age, a container box/ Memory Location is named Age and the value 20 is stored in the memory address 1000245 with name/label, Age. Now, on reassigning the value 30 to Age, the value 30 is stored in the same memory location. This is how the variables behave in Traditional programming languages.

==========

Internal mechanism of variable(Important Concept)

What is actually happening when you make a variable assignment? This is an important question in Python, because the answer differs somewhat from what you'd find in many other programming languages.
Python is a highly object-oriented language. In fact, virtually every item of data in a Python program is an object of a specific type or class.
Consider this code:

In [11]:
```
print(300)
```

```
300
```

When presented with the statement **print(300)**, the interpreter does the following:

- Creates an integer object
- Gives it the value 300
- Displays it to the console

You can see that an integer object is created using the built-in type() function:

In [12]:
```
type(300)
```

Out[12]:
```
int
```

A Python variable is a symbolic name that is a reference or pointer to an object. Once an object is assigned to a variable, you can refer to the object by that name. But the data itself is still contained within the object.
For example:

In [13]:
```
n=300
```

Variable Assignment

The following code verifies that n points to an integer object:

In [14]:
```
print(n)

type(n)
```

```
300
```

Out[14]:
```
int
```

Now consider the following statement:

In [15]:
```
m=n
```

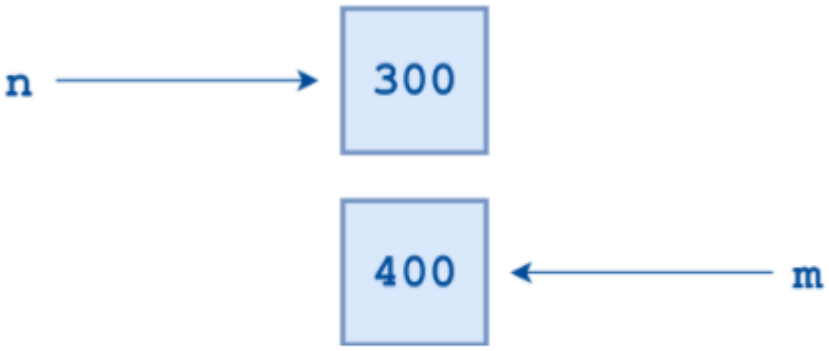
What happens when it is executed? Python does not create another object. It simply creates a new symbolic name or reference, m, which points to the same object that n points to.

Multiple References to a Single Object

Next, suppose you do this:

In [16]:
```
m=400
```

Now Python creates a new integer object with the value 400, and m becomes a reference to it.

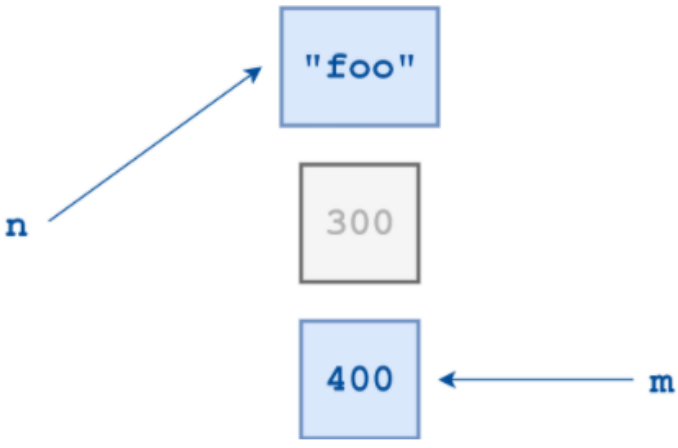


References to Separate Objects

Lastly, suppose this statement is executed next:

In [17]:
```
n = "foo"
```

Now Python creates a string object with the value "foo" and makes n reference that.



Orphaned Object

==========

Object identity

In [18]:
```
Name = "Sachin"
Nickname = "Sachin"
print(Name)
print(Nickname)
```

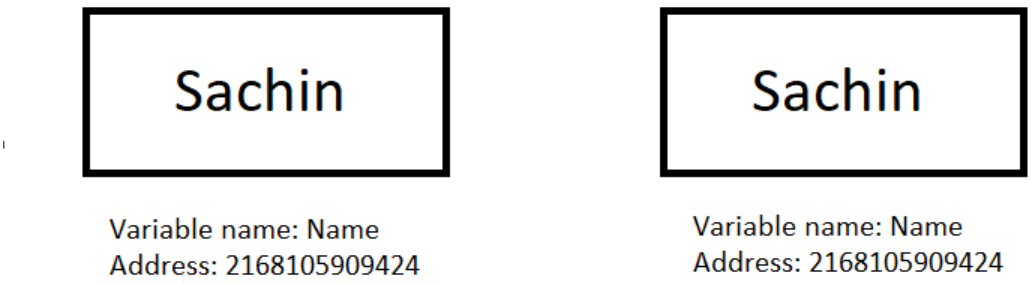
```
Sachin
Sachin
```

As you can see in above program i declared two variable first one is name assigned with "Sachin" and other variable name is Nickname also assigned with "Sachin". If you think there are two different variable so they both have two different memory address. let's see the address of these two variables.

Important point

In [19]:
```
print(id(Name))
print(id(Nickname))
```

```
2310378658416
2310378658416
```



Why they both have the same memory address??????

Ooops Name and Nickname both have the same memory address. This is the beauty of Python. If u have any number of variables and you assign with the same content. so in memory a space is occupied by "sachin" and all the variable is pointing to same memory address.

==========

Interesting hack:

Interesting.

After I did some experimentation, it appears that when you assign a name to an integer between -5 to 256 inclusively, Python assigns predetermined memory locations that contain that integer to the name. For numbers over 256, Python seems to create a new memory location and fill it with the desired number, before pointing the name to that memory location.

I suspect that this behavior is unique to python and isn't something you should rely on as it probably works differently in other Python implementations.

In [20]:
```
X=-7
Y=-7

print("This print is of -7")
print(id(X))
print(id(Y))
print("*********************************************************")
X=-6
Y=-6

print("This print is of -6")
print(id(X))
print(id(Y))
print("*********************************************************")
X=-5
Y=-5

print("This print is of -5")
print(id(X))
print(id(Y))
print("*********************************************************")
X=-4
Y=-4

print("This print is of -4")
print(id(X))
print(id(Y))
print("*********************************************************")
X=0
Y=0

print("This print is of 0")
print(id(X))
print(id(Y))
print("*********************************************************")
X=10
Y=10

print("This print is of 10")
print(id(X))
print(id(Y))
print("*********************************************************")
X=100
Y=100
```

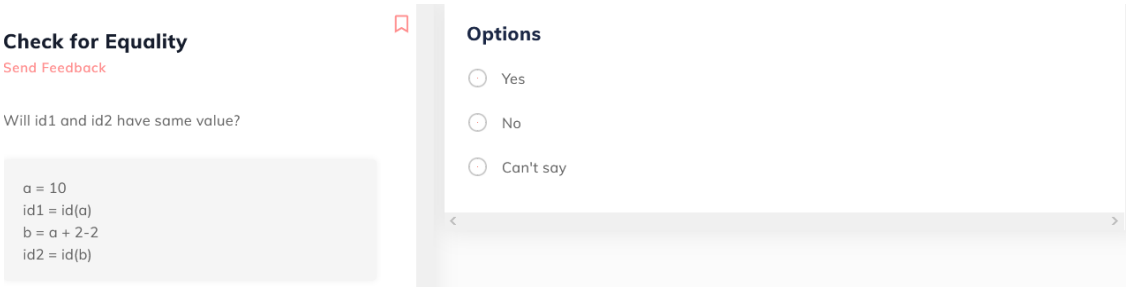
```
print("This print is of 100")
print(id(X))
print(id(Y))
print("*****************************************************")
X=255
Y=255

print("This print is of 255")
print(id(X))
print(id(Y))
print("*****************************************************")
X=256
Y=256

print("This print is of 256")
print(id(X))
print(id(Y))
print("*****************************************************")
X=257
Y=257

print("This print is of 257")
print(id(X))
print(id(Y))
print("*****************************************************")
X=300
Y=300

print("This print is of 300")
print(id(X))
print(id(Y))
```

```
This print is of -7
2310378973456
2310378974832
*****************************************************
This print is of -6
2310378974416
2310378973648
*****************************************************
This print is of -5
140709241431648
140709241431648
*****************************************************
This print is of -4
140709241431680
140709241431680
*****************************************************
This print is of 0
140709241431808
140709241431808
*****************************************************
This print is of 10
140709241432128
140709241432128
*****************************************************
This print is of 100
140709241435008
140709241435008
*****************************************************
This print is of 255
140709241439968
140709241439968
*****************************************************
This print is of 256
140709241440000
```

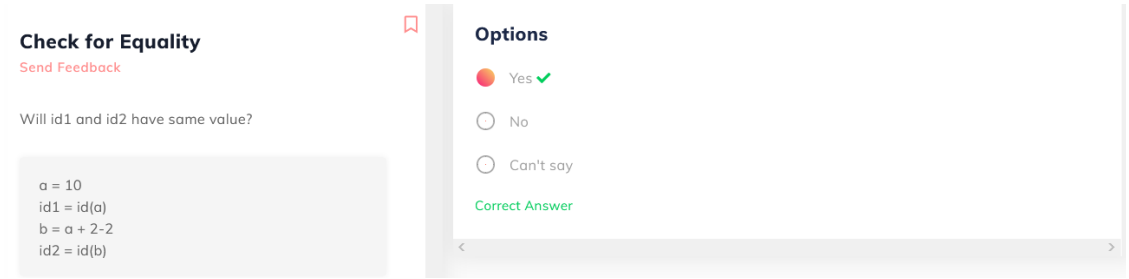
```
140709241440000
**************************************************
This print is of 257
2310379360592
2310379361040
**************************************************
This print is of 300
2310379360752
2310379361200
```

Code Explanation: As you can see from the above code for variable -4,0,10,100,255,256 for this the python store on same memory location because this range lies between -5 to 256. So, python use predetermined storage. And for value which is not lie between -5 to 256 lie between like -7,-6,300 python store it on different different place.

Quiz



Solution



==========

Data types:

Variables can hold values, and every value has a data-type. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

In [21]:
```
var=10
```

The variable "Var" holds integer value ten and we did not define its type. Python interpreter will automatically interpret variable Var as an integer type.
Python enables us to check the type of the variable used in the program. Python provides us the **type()** function, which returns the type of the variable passed.
Consider the following example to define the values of different data types and checking its type.

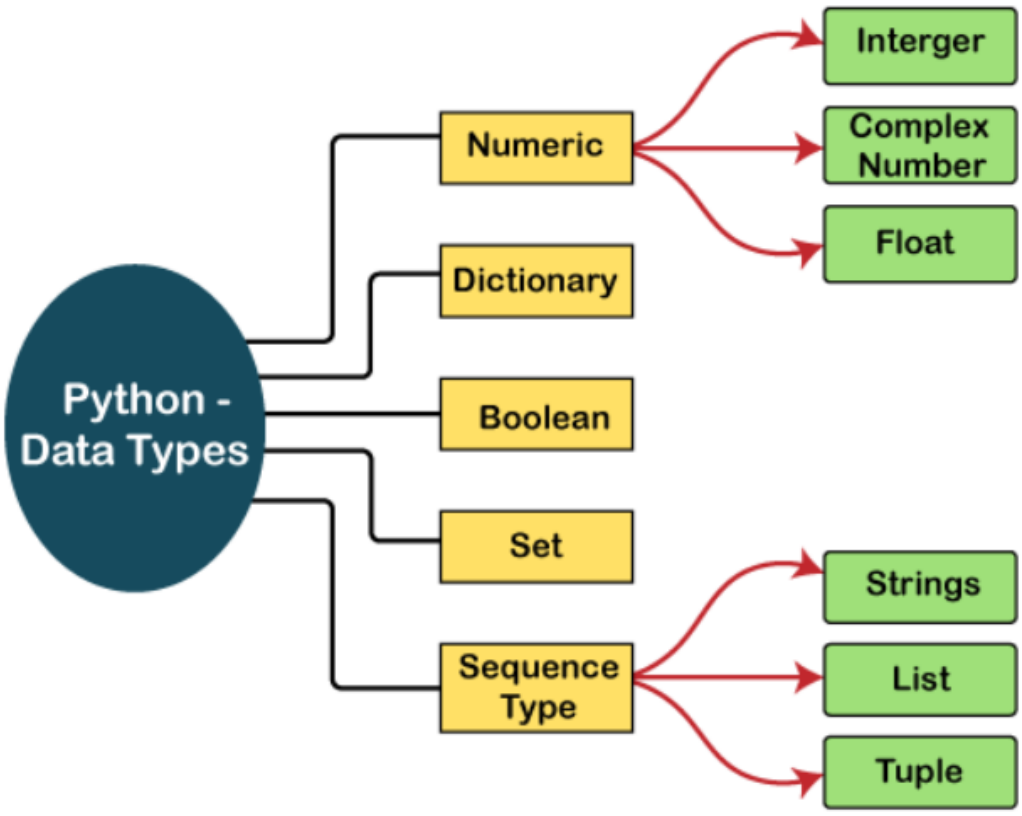
In [22]:
```
var1=10
```

```
var2="Hi Python"
var3=10.5
print(type(var1))
print(type(var2))
print(type(var3))
```

```
<class 'int'>
<class 'str'>
<class 'float'>
```

Standard Data Type: A variable can hold different types of values. For example, a person's name must be stored as a string where as its id must be stored as an integer. Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

1. Number
2. Sequence type
3. Boolean
4. Set
5. Dictionary



1. Numbers:

Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type. Python provides the **type()** function to know the data-type of the variable. Similarly, the **isinstance()** function is used to check an object belongs to a particular class.

Python creates Number objects when a number is assigned to a variable. For example;

Example:

In [23]:
```
a = 5
print("The type of a", type(a))

b = 40.5
print("The type of b", type(b))

c = 1+3j
print("The type of c", type(c))

print(" c is a complex number", isinstance(1+3j,complex))
```

```
The type of a <class 'int'>
The type of b <class 'float'>
The type of c <class 'complex'>
 c is a complex number True
```

Python supports three types of numeric data.

1. **Int** - Integer value can be any length such as integers 10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer. Its value belongs to int.
2. **Float** - Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.
3. **complex** - A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

2. Sequence Type:

String: The string can be defined as the sequence of characters represented in the quotation marks. In Python, we can use single, double, or triple quotes to define a string.

String handling in Python is a straight forward task since Python provides built-in functions and operators to perform operations in the string.

Example:

In [24]:
```
str = "string using double quotes"
print(str)

s = '''''A multiline
string'''

print(s)
```

```
string using double quotes
''A multiline
string
```

List: Lists are similar to arrays in C. However, the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

Example:

In [25]:
```
list1  = [1, "hi", "Python", 2]
#Checking type of given list
print(type(list1))

#Printing the list1
print (list1)
```

```
<class 'list'>
[1, 'hi', 'Python', 2]
```

Tuple: A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

Example:

In [26]:
```
tup  = ("hi", "Python", 2)
# Checking type of tup
print (type(tup))

#Printing the tuple
print (tup)
```

```
<class 'tuple'>
('hi', 'Python', 2)
```

3. Dictionary:

Dictionary is an unordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type, whereas value is an arbitrary Python object.

The items in the dictionary are separated with the comma (,) and enclosed in the curly braces {}.

Example:

In [27]:
```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'}

# Printing dictionary
print (d)
```

```
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
```

4. Boolean:

Boolean type provides two built-in values, True and False. These values are used to determine the given statement true or false. It denotes by the class bool. True can be represented by any non-zero value or 'T' whereas false can be represented by the 0 or 'F'.

Example:

In [28]:
```
# Python program to check the boolean type
print(type(True))
print(type(False))
print(false) #First letter should be capital
```

```
<class 'bool'>
<class 'bool'>
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-28-3de3c5240371> in <module>
      2 print(type(True))
      3 print(type(False))
```

```
----> 4 print(false) #First letter should be capital

NameError: name 'false' is not defined
```

5. Set:

Set is the unordered collection of the data type. It is iterable, mutable(can modify after creation), and has unique elements. In set, the order of the elements is undefined; it may return the changed sequence of the element. The set is created by using a built-in function set(), or a sequence of elements is passed in the curly braces and separated by the comma. It can contain various types of values.

Example:

In [29]:
```
# Creating Empty set
set1 = set()

# Creating set with elements
set2 = {'James', 2, 3,'Python'}

#Printing Set value
print(set2)
```

```
{3, 'James', 2, 'Python'}
```

Getting the Data type:

You can get the data types of any type by using **type()** function

Example:

In [30]:
```
x=5
print("Data type of x is: ",type(x))

x="Sachin Kapoor"
print("Data type of x is: ",type(x))
```

```
Data type of x is:  <class 'int'>
Data type of x is:  <class 'str'>
```

Data type conversion:

you can convert from one type to another with the int(), float(), str() and complex() and maany more method.

Example:

In [31]:
```
x=5
print("Data type of x is: ",type(x))

x=float(5)
print("Data type of x after type conversion is: ",type(x))
```

```
Data type of x is:  <class 'int'>
Data type of x after type conversion is:  <class 'float'>
```

Explore via coding:

Write a program to input marks of three tests of a student (all integers). Then

calculate and print the average of all test marks.

In [32]:
```
sub1=int(input("Enter the first subject marks: "))
sub2=int(input("Enter the second subject marks: "))
sub3=int(input("Enter the third subject marks: "))

total_marks=sub1+sub2+sub3
avg_marks=total_marks/3
print("Average marks of the student is: ",avg_marks)
```

```
Enter the first subject marks: 25
Enter the second subject marks: 40
Enter the third subject marks: 60
Average marks of the student is:  41.666666666666664
```

Write a python program to find the area of rectngle.

In [33]:
```
l=int(input("Enter the length of rectangle: "))
w=int(input("Enter the length of rectangle: "))

area=l*w
print("The area of rectangle is: ",area)
```

```
Enter the length of rectangle: 38
Enter the length of rectangle: 12
The area of rectangle is:  456
```

==========

Taking Input From User

Developers often need to interact with users, either to get data or to provide some sort of result. To get the input from the user interactively, we can use the built-in function, input(). This function is used in the following manner:

Syntax:

In [ ]:
```
variable_to_hold_the_input_value = input(Prompt to be displayed)
```

Example:

In [34]:
```
age = input("What is your age: ")
```

```
What is your age: sachin kapoor
```

Example:

In [35]:
```
name = input("Enter your name: ")
age = input("Enter your age: ")
print(name)
print(age)
```

```
Enter your name: sachin kapoor
Enter your age: 22
sachin kapoor
22
```

Note: input() function always returns a value of the String type. Notice that in the above script the output for both name and age, Python has enclosed the output in quotes, like 'sachin kapoor' and '22', which implies that it is of String type. This is just because, whatever the user inputs in the input() function, it is treated as a String. This would mean

that even if we input an integer value like 22, it will be treated like a string '22' and not an integer. Now, we will see how to read Numbers in the next section.

Reading Numbers

Python offers two functions int() and float() to be used with the input() function to convert the values received through input() into the respective numeric types integer and

floating-point numbers. The steps will be:-

1. Use the input() function to read the user input.
2. Use the int() and float() function to convert the value read into integers and floating-point numbers, respectively. This process is called Type Casting.

The general way of taking Input:

Syntax:

In [ ]:
```
variableRead = input(<Prompt to be displayed>)<br>
updatedVariable = int(variableRead)
```

Here, variableRead is a String type that was read from the user. This string value will then be converted to Integer using the int() function and assigned to updatedVariable.
This can even be shortened to a single line of code as shown below:-

Syntax:

In [ ]:
```
updatedVariable = int(input(<Prompt to be displayed>))
```

Example:

In [36]:
```
age= int(input("Enter Your Age: "))
print(age)
```

```
Enter Your Age: 22
22
```

==========

Keywords:

Keywords are the reserved words in python. we can't use use a keyword as a variable name, function name or any other identifier</p>

Here's a list of all keywords in Python Programming

In [37]:
```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'brea
k', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finall
y', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlo
cal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yi
eld']
```

Note:

- Total 35 keywords.
- Except True and False all keyword is in lower case.

==========

Identifiers:

Identifier An identifier is a name given to entity like Class, Functions, Variables etc. It helps to differentiate one entity from another</p>

Rules for writing identifier

1. Identifiers can be a combinations of letter in Lowercase(a-z) or uppercase(A-Z) or digits (0-9) or an underscore
2. An identifier can't start with a digit is invalid
3. We can't use special symbol

==========

Python comment

- Comments can be used to explain Python code
- Comments can be used to make the code more redable
- comments can be used to prevent execution while testing code
- Comments starts with a "#" Python will ignore them
- Comments can be placed at the end of a line, and Python will ignore the rest of the line

Multiple Line comment:

Python does not really have a syntax for multiline comments.

To add a multiline comment you could insert a "#" for each line:

or, not quite as intended, you can use a multiline string.

NOTE: Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string(triple quote) in your code and place your comment inside it

==========

Mutable and Immutable Objects:

Data objects of the above types are stored in a computer's memory for processing. Some of these values can be modified during processing, but contents of others can't be altered once they are created in the memory.

Number values, strings, and tuple are immutable, which means their contents can't be altered after creation.

On the other hand, collection of items in a List or Dictionary object can be modified. It is possible to add, delete, insert, and rearrange items in a list or dictionary. Hence, they are mutable objects.

NOTE: We will discuss this later in this notebook in depth.

==========

Python operators:

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Identity operators
6. Membership operators
7. Bitwise operators

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

In [ ]:
```
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
```

Example of Arithmetic operator

In [38]:
```
X = 10
Y = 2

print("Result of addition",X+Y)
print("Result of subtraction",X-Y)
print("Result of multiplication",X*Y)
print("Result of division",X/Y)
print("Result of modulus",X%Y)
print("Result of exponentiation",X**Y)
print("Result of floor division",X//Y)
```

```
Result of addition 12
Result of subtraction 8
Result of multiplication 20
Result of division 5.0
Result of modulus 0
Result of exponentiation 100
Result of floor division 5
```

Assignment Operators: Assignment operators are used to assign values to variables:

In [ ]:
```
Operator                Example             Same as

   =                      x=5                 x=5
  +=                     x+=5                x=x+5
  -=                     x-=5                x=x-5
```

*=	x*=5	x=x*5
/=	x/=5	x=x/5
%=	x%=5	x=x%5
**=	x**=5	x=x**5
//=	x//=5	x=x//5
&=	x&=3	x=x&3
|=	x|=3	x=x|3
>>=	x>>3	x=x>>3
<<=	x<<=3	x=x<<3

Example of Assignment operator

In [39]:
```
X = 10

print("Value of X is",X)
x=10
X+=5
print("Result of assignment addition",X)
X=10
X-=5
print("Result of assignment subtraction",X)
X=10
X*=5
print("Result of assignment multiplication",X)
X=10
X/=5
print("Result of assignment division",X)
X=10
X%=5
print("Result of assignment modulus",X)
X=10
X**=5
print("Result of assignment exponentiation",X)
X=10
X//=5
print("Result of assignment floor division",X)
X=10
X&=5
print("Result of and operator",X)
X=10
X|=5
print("Result of Or operator",X)
X=10
X>>=5
print("Result of assignment right shift",)
X=10
X<<=5
print("Result of assignment left shift",X)
```

```
Value of X is 10
Result of assignment addition 15
Result of assignment subtraction 5
Result of assignment multiplication 50
Result of assignment division 2.0
Result of assignment modulus 0
Result of assignment exponentiation 100000
Result of assignment floor division 2
Result of and operator 0
Result of Or operator 15
Result of assignment right shift
Result of assignment left shift 320
```

Comparison Operators: Comparison operators are used to compare two values:

In [ ]:
```
Operator                    Name                    Example
```

```
==                      Equal                       X==Y
!=                      Not equal                   X!=Y
>                       Greater than                X>Y
<                       Less than                   X<Y
>=                      Greater than                X>=Y
<=                      Less than                   X<=Y
```

Example of Comparison operator

In [40]:
```
X=10
Y=5

print("Result of equal",X==Y)
print("Result of not equal",X!=Y)
print("Result of greater than",X>Y)
print("Result of less than",X<Y)
print("Result of greater than equal",X>=Y)
print("Result of less than equal",X<=Y)
```

```
Result of equal False
Result of not equal True
Result of greater than True
Result of less than False
Result of greater than equal True
Result of less than equal False
```

Logical Operators: Logical operators are used to combine conditional statements:

In [ ]:
```
Operator                        Description

   and              Returns True if both statements are true
   or               Returns True if one of the statements is true
   not              Reverse the result, returns False if the result is t
```

Example of Logical operator

In [41]:
```
x=10
y=5
z=2

print("Result of and",(x>y and x>z))
print("Result of or",(x>y or x>z))
print("Result of not",not(x>y and x>z))
```

```
Result of and True
Result of or True
Result of not False
```

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

In [ ]:
```
Operator                        Description

   in           Returns True if a sequence with the specified value is pre
   not in       Returns True if a sequence with the specified value is not
```

Example of Membership operator

In [42]:
```
x = ["apple", "banana"]

print("banana" in x)

# returns True because a sequence with the value "banana" is in the list
```

```
True
```

In [43]:
```
x = ["apple", "banana"]

print("pineapple" not in x)

# returns True because a sequence with the value "pineapple" is not in t
```

```
True
```

Bitwise Operators : Bitwise operators are used to compare (binary) numbers:

In [ ]:
```
Operator      Name                                                Descript
  &           AND                                 Sets each bit to 1 if both b
  |           OR                                  Sets each bit to 1 if one of
  ^           XOR                                 Sets each bit to 1 if only o
  ~           NOT                                 Inverts all the bits
  <<          Zero fill left shift                Shift left by pushing zeros
  >>          Signed right shift                  Shift right by pushing copie
                                                  rightmost bits fall off
```

==========

Precedence of python operator:

The combination of values, variables, operators and function call.

The combination of values, variables and operator and function call is termed as expression. The Python interpreter can evaluate a valid expression.

Example

In [44]:
```
5-7
```

Out[44]:
```
-2
```

Here 5-7 is an expression. There can be more than one operator in an expression.

To evaluate these types of expression there is a rule of precedence in Python. It guides the order in which these operations are carried out

for example multiplication has higher precedence than subtraction.

In [45]:
```
10-4*2
```

Out[45]:
```
2
```

But we can change this order by parenthesis.

In [46]:
```
(10-4)*2
```

Out[46]:
```
12
```

The operator precedence in Python is listed below. It is in descending order(upper group has higher precedence than the lower ones)

In [ ]:
```
Operators                                                   Meaning

()                                                      Parentheses
**                                                      Exponent
```

```
+x, -x, ~x                                          Unary plus, U
*, /, //, %                                         Multiplicatio
+, -                                                Addition, Sub
<<, >>                                              Bitwise shift
&                                                   Bitwise AND
^                                                   Bitwise XOR
|                                                   Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in        Comparisons,
not                                                 Logical NOT
and                                                 Logical AND
or                                                  Logical OR
```

Associativity of Python operator:

We can see in the above table that more than one operator exist in the same group. These operator have the same precedence.

when two operators have the same precedence, associativity helps to determine the order of operations.

Associativity is the order in which an expression is evaluated that has multiple operators of the same precedence. Almost all the operators have the left to right associativity.

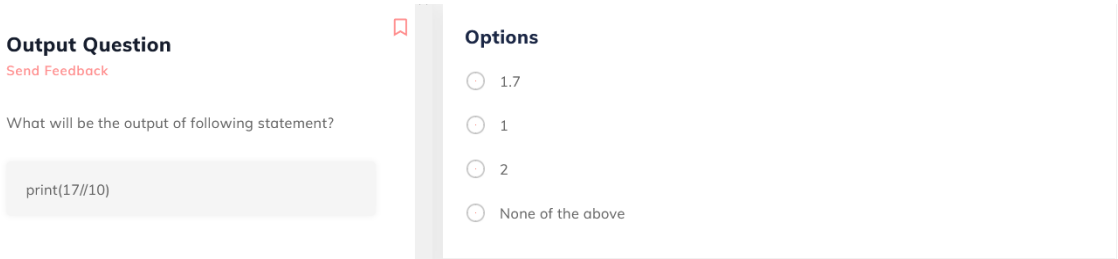
For example: multiplication and floor division have the same precedence. Hence if both of them are present in an expression, the left one is evaluated first.

Non-associative operators:

Some operators like assignment and comparison operators do not have associativity in Python. There are seperator rules for sequence of this kind of operator and can not be expressed as associativity.

For example: x<y<z neither means (x<y)<z nor x<(y<z). x<y<z is equivalent to x<y and y<z and is evaluated from left to right.

Quiz



Solution

Output Question

Send Feedback

What will be the output of following statement?

```
print(17//10)
```

Options

- 1.7
- 1 ✔
- 2
- None of the above

Correct Answer

Quiz

Output Question

Send Feedback

What will be the output of following statement?

```
print(17/10)
```

Options

- 1.7
- 1
- 2
- None of the above

Solution

Output Question

Send Feedback

What will be the output of following statement?

```
print(17/10)
```

Options

- 1.7 ✔
- 1
- 2
- None of the above

Correct Answer

Quiz

Output Question

Send Feedback

What will be the output of the code if input provided is 40 and 57 ?

```
a = input()
b= input()
C = a+b
print(C)
```

Options

- 97
- "40+57"
- "4057"
- None of the above

Solution

Output Question

Send Feedback

What will be the output of the code if input provided is 40 and 57 ?

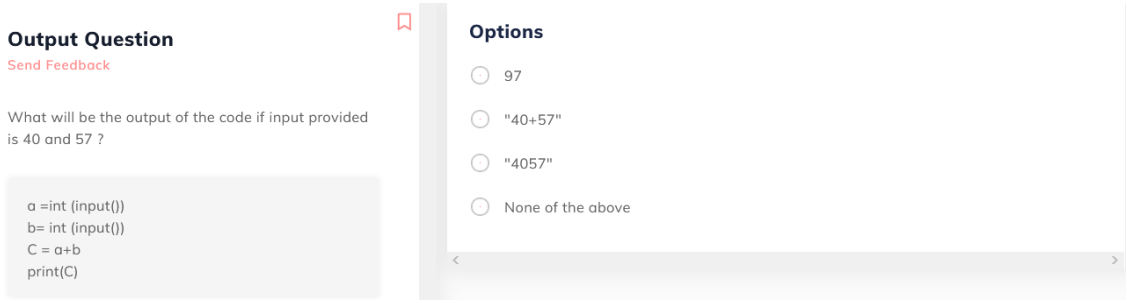
```
a = input()
b= input()
C = a+b
print(C)
```

Options

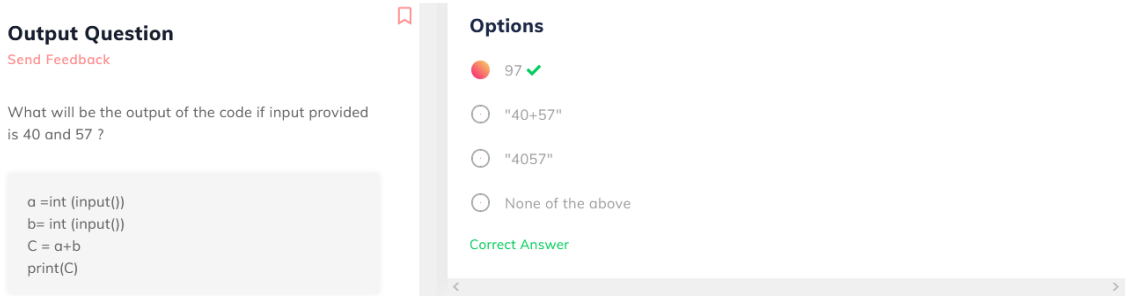
- 97
- "40+57"
- "4057" ✔
- None of the above

Correct Answer

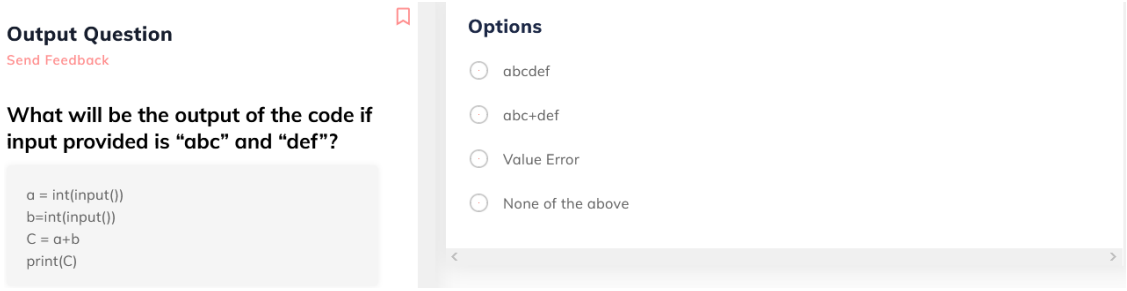
Quiz



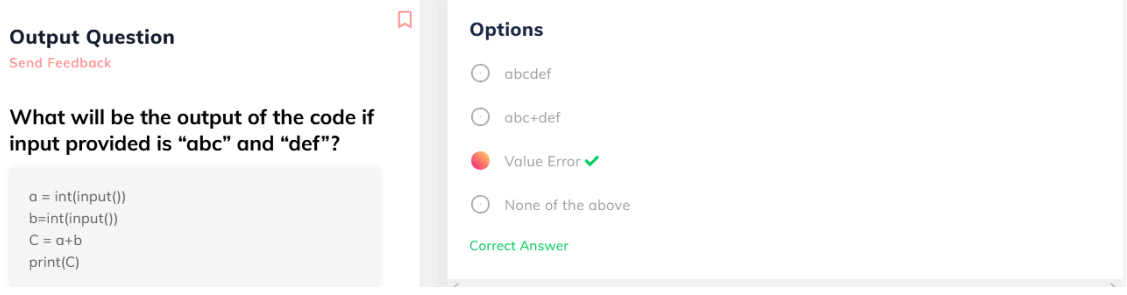
Solution



Quiz



Solution



==========

Conditional statements:

There are certain points in our code when we need to make some decisions and then based on the outcome of those decisions we execute the next block of code. Such conditional statements in programming languages control the flow of program execution.

Most commonly used conditional statements in Python are:

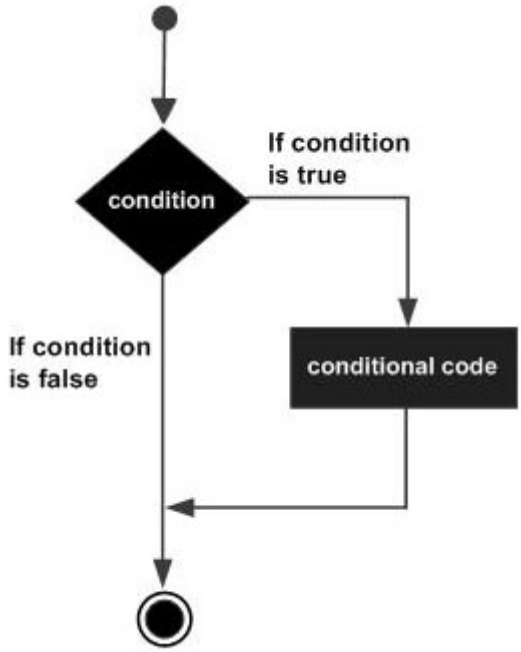
- Simple If statements
- If-Else statements
- If-Elif statements
- Nested Conditionals

If statement:

These are the most simple decision-making/conditional statements. It is used to decide whether a certain statement or block of statements will be executed or not.

- The most important part of any conditional statement is a condition or a boolean.
- And the second important thing is the code block to be executed.

In the case of simple If statements, if the conditional/boolean is true then the given code block is executed, else the code block is simply skipped and the flow of operation comes out of this If condition.



Syntax of if Statement:

In [ ]:
```
if (test expression):
    statement(s)
```

Here, the program evaluates the test expression and will execute statement(s) only if the test expression is True.

If the test expression is False, the statement(s) is not executed.

In Python, the body of the if statement is indicated by the indentation. The body starts with an indentation and the first unindented line marks the end.

Python interprets non-zero values as True. None and 0 are interpreted as False.

Example:

In [47]:
```
num=3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")

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

```
3 is a positive number.
This is always printed.
This is also always printed.
```

Code Explanation:

In the above example, num > 0 is the test expression.
The body of if is executed only if this evaluates to True.
When the variable num is equal to 3, test expression is true and statements inside the body of if are executed.
If the variable num is equal to -1, test expression is false and statements inside the body of if are skipped.
The print() statement falls outside of the if block (unindented). Hence, it is executed regardless of the test expression.

Explore via coding:

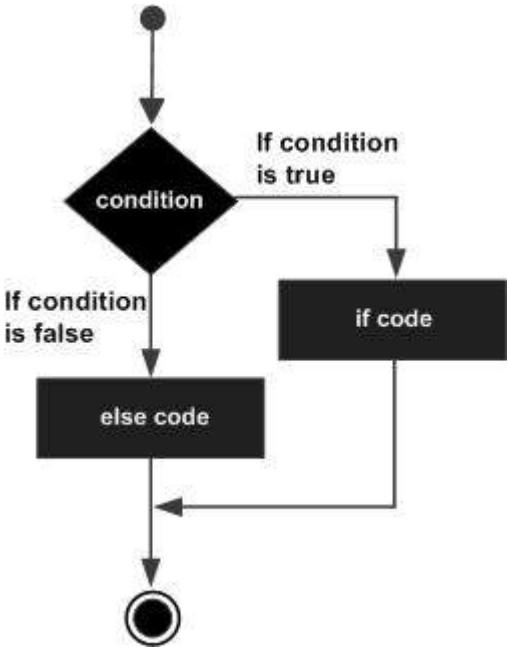
Write a python program to input a variable from user and check wheather user enter integer variable or not.

In [48]:
```
var=int(input("Enter the variable: "))

if(isinstance(var,int)):
    print("Yes it is a integer variable")
```

```
Enter the variable: 23
Yes it is a integer variable
```

if...else Statement

The simple if statement, tells us that if a condition is true it will execute a block of statements, and if the condition is false it won't. But what if we want some other block of code to be executed if the condition is false. Here comes the else statement. We can use the else statement with if statement to execute a block of code when the condition is false.



Syntax of if... else:

In [ ]:
```
if (test expression):
    statement(s)
else:
    statement(s)
```

The if..else statement evaluates test expression and will execute the body of if only when the test condition is True.

If the condition is False, the body of else is executed. Indentation is used to separate the blocks.

Example

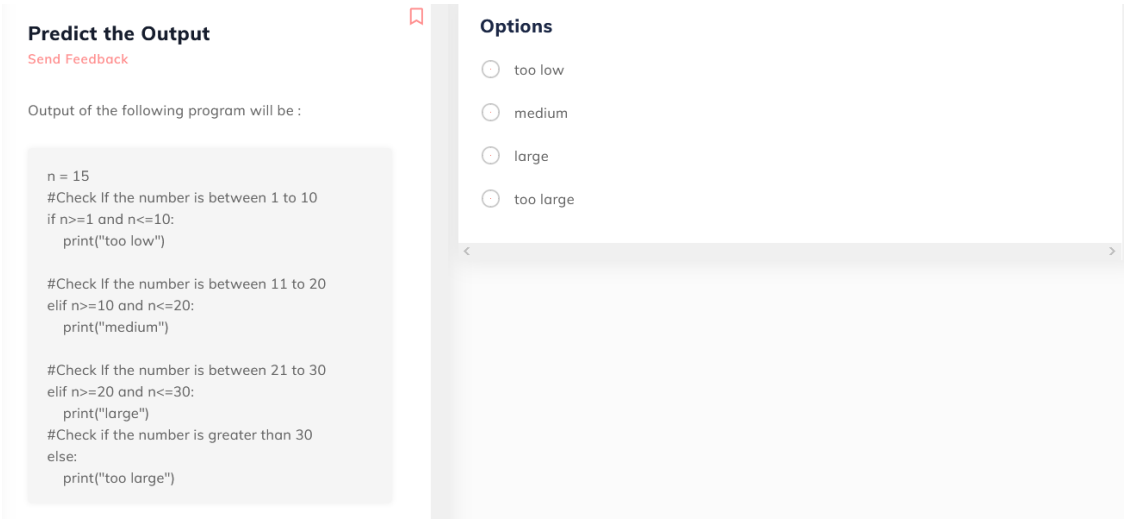
In [49]:
```
num = 3

if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

```
Positive or Zero
```

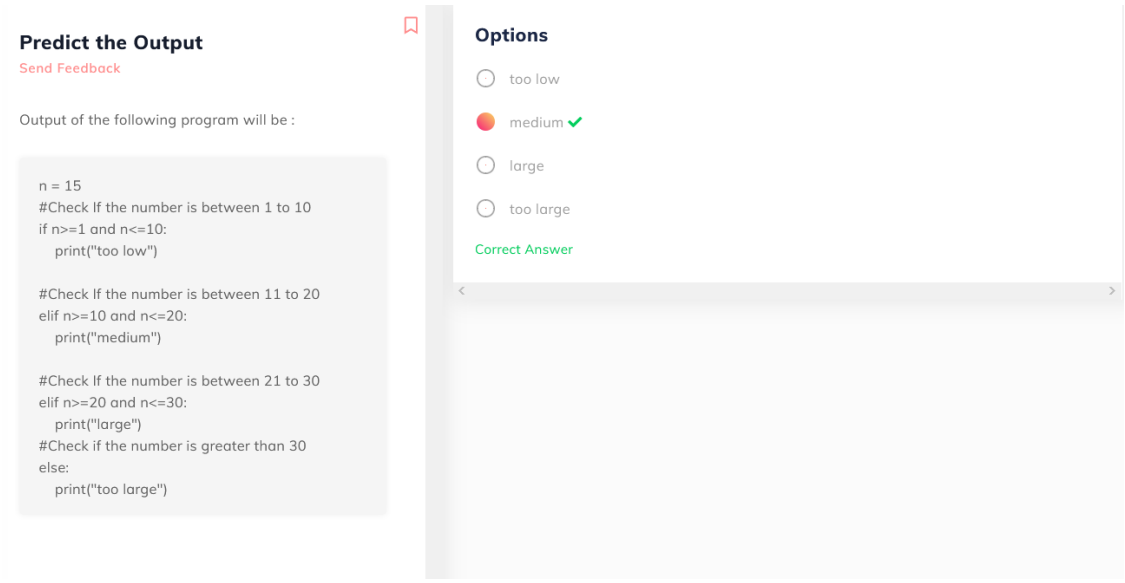
Code Explanation:

In the above example, when num is equal to 3, the test expression is true and the body of if is executed and the body of else is skipped.
If num is equal to -5, the test expression is false and the body of else is executed and the body of if is skipped.
If num is equal to 0, the test expression is true and body of if is executed and body of else is skipped.

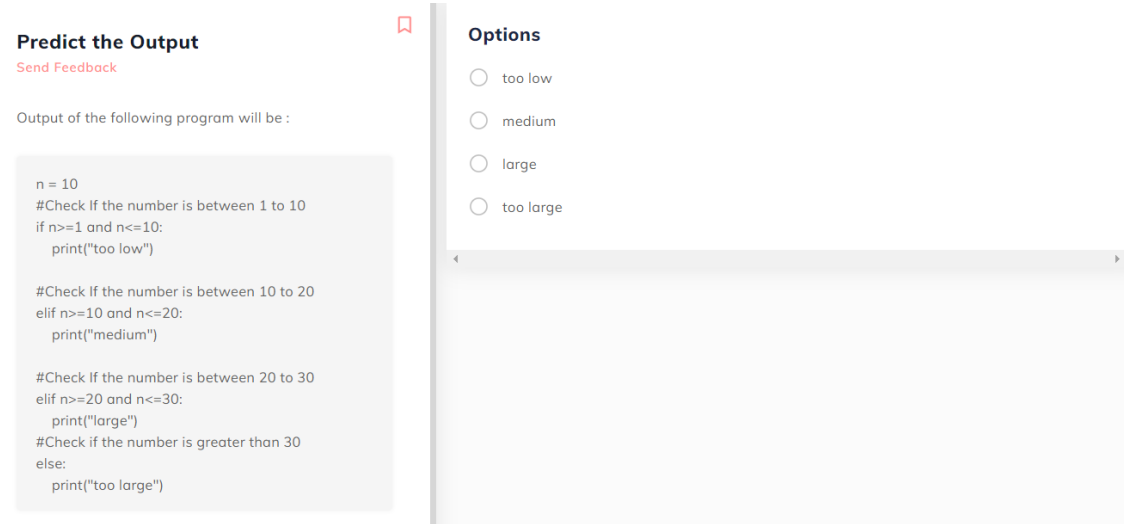
Quiz



Solution



Quiz



Solution

Predict the Output

Send Feedback

Output of the following program will be :

```
n = 10
#Check If the number is between 1 to 10
if n>=1 and n<=10:
   print("too low")

#Check If the number is between 10 to 20
elif n>=10 and n<=20:
   print("medium")

#Check If the number is between 20 to 30
elif n>=20 and n<=30:
   print("large")
#Check if the number is greater than 30
else:
   print("too large")
```

Options

- too low ✔
- medium
- large
- too large

Correct Answer

Quiz

Figure out the output

Send Feedback

What will the following code segment print?

```
x = 15
if x <= 15:
   print("Inside if")
else:
   print("Inside else")
```

Options

- Inside If
- Inside else
- Inside If Inside else

Solution

Figure out the output

Send Feedback

What will the following code segment print?

```
x = 15
if x <= 15:
   print("Inside if")
else:
   print("Inside else")
```

Options

- Inside If ✔
- Inside else
- Inside If Inside else

Correct Answer

Quiz

Multiple Ifs

Send Feedback

Consider the following piece of code -

```
x = 5
if x < 6:
   print("Hello")
if x == 5:
   print("Hi")
else:
   print("Hey")
```

Which of the above 3 print statement(s) will be executed?

Options

- [ ] print("Hello")
- [ ] print("Hi")
- [ ] print("Hey")
- [ ] All 3 will execute

Solution

Multiple Ifs

Send Feedback

Consider the following piece of code -

```
x = 5
if x < 6:
    print("Hello")
if x == 5:
    print("Hi")
else:
    print("Hey")
```

Which of the above 3 print statement(s) will be executed?

Options

- [ ] print("Hello")
- [x] print("Hi")
- [ ] print("Hey")
- [ ] All 3 will execute

Wrong Answer, Attempt Again

Quiz

Conditional Question

Send Feedback

What will to following code segment print?

```
if (10 < 0) and (0 < -10):
    print("A")
elif (10 > 0) or False:
    print("B")
else:
    print("C")
```

Options

- ( ) A
- ( ) B
- ( ) C
- ( ) B & C

Solution

Conditional Question

Send Feedback

What will to following code segment print?

```
if (10 < 0) and (0 < -10):
    print("A")
elif (10 > 0) or False:
    print("B")
else:
    print("C")
```

Options

- ( ) A
- (•) B ✓
- ( ) C
- ( ) B & C

Correct Answer

Quiz

Conditional Question

Send Feedback

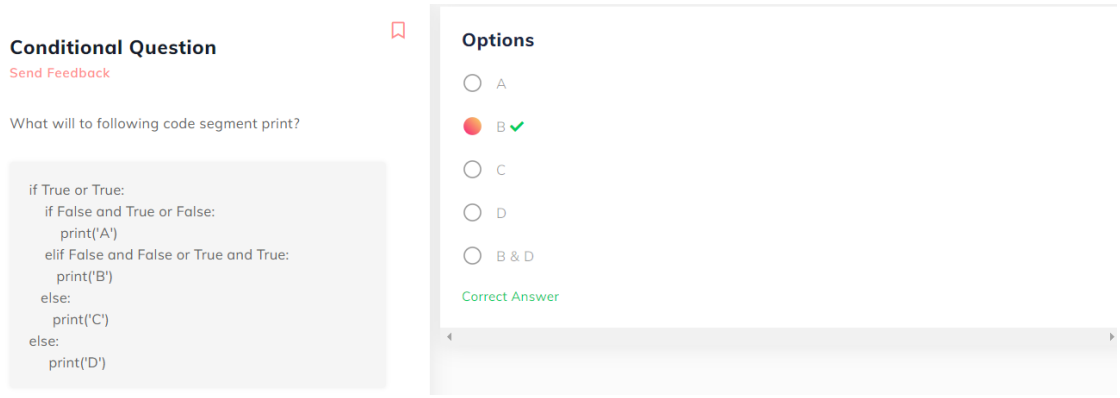
What will to following code segment print?

```
if True or True:
    if False and True or False:
        print('A')
    elif False and False or True and True:
        print('B')
    else:
        print('C')
else:
    print('D')
```

Options

- ( ) A
- ( ) B
- ( ) C
- ( ) D
- ( ) B & D

Solution



Explore via coding:

Write a python program to take age as a input and check if his/her age is greater than 18 sohe/she is eligible for voting if age is less than 18 he/she is not eligible for voting.

In [50]:
```
age=int(input("Enter the age: "))

if(age>18):
    print("You are eligible for voting")
else:
    print("You are not eligible for voting")
```

```
Enter the age: 22
You are eligible for voting
```

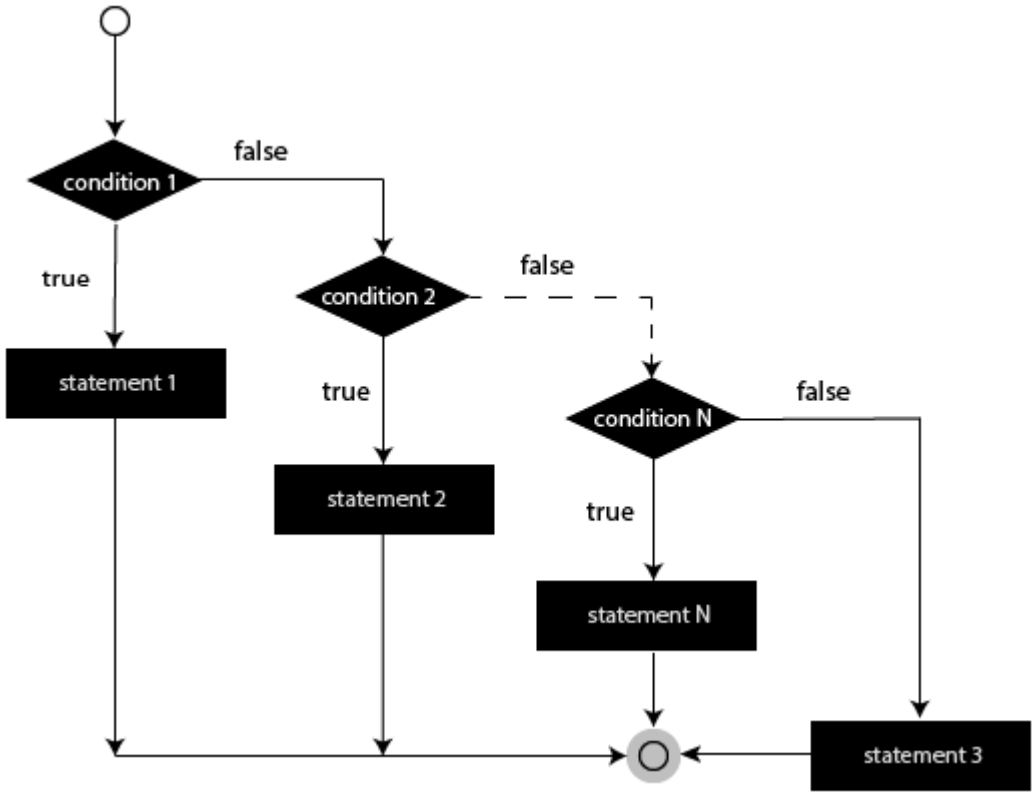
Write a program to take a number from user and check number is prime or not.

In [51]:
```
n=int(input("enter the number: "))

if(n%2==0):
    print("It is even number")
else:
    print("It is odd number")
```

```
enter the number: 17
It is odd number
```

If.. elif.. else statement

So far we have looked at Simple If and a single If-Else statement. However, imagine a situation in which if a condition is satisfied, we want a particular block of code to be executed, and if some other condition is fulfilled we want some other block of code to run. However, if none of the conditions is fulfilled, we want some third block of code to be executed. In this case, we use an if-elif-else ladder. In this, the program decides among multiple conditionals. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.



Syntax of if...elif...else

In [ ]:
```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

The elif is short for else if. It allows us to check for multiple expressions.

If the condition for if is False, it checks the condition of the next elif block and so on.

If all the conditions are False, the body of else is executed.

Only one block among the several if...elif...else blocks is executed according to the condition.

The if block can have only one else block. But it can have multiple elif blocks.

Note: We can have as many elif statements as we want, between the if and the else statements. This means we can consider as many conditions as we want. It should be noted that once an if or elif condition is executed, the remaining elif and else statements will not be executed.

Example:

In [52]:
```
num = 3.4

if num > 0:
    print("Positive number")
```

```
elif num == 0:
    print("Zero")

else:
    print("Negative number")
```

```
Positive number
```

Code Explanation:

When variable num is positive, Positive number is printed.

If num is equal to 0, Zero is printed.

If num is negative, Negative number is printed.

Explore via coding:

Write a python program to build a calculator take two operand and one operator and perform function as per operator.

In [54]:
```
while(True):
    opr1=int(input("Enter the first operand: "))
    opr2=int(input("Enter the second operand: "))
    opr=input("Enter the operator: ")

    if(opr=="+"):
        print("Addition of two number is: ",opr1+opr2)
    elif(opr=="-"):
        print("Subtraction of two number is: ",opr1-opr2)
    elif(opr=="*"):
        print("Multiplication of two number is: ",opr1*opr2)
    elif(opr=="/"):
        print("Division of two number is: ",opr1/opr2)
    print("****************************************")
    print()
    choice=input("You want to continue Y/N")
    if(choice[0]=="N"):
        break
```

```
Enter the first operand: 10
Enter the second operand: 20
Enter the operator: +
Addition of two number is:  30
****************************************

You want to continue Y/NY
Enter the first operand: 20
Enter the second operand: 10
Enter the operator: -
Subtraction of two number is:  10
****************************************

You want to continue Y/N5
Enter the first operand: 5
Enter the second operand: 5
Enter the operator: *
Multiplication of two number is:  25
****************************************

You want to continue Y/NY
Enter the first operand: 20
Enter the second operand: 2
Enter the operator: /
Division of two number is:  10.0
```

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

You want to continue Y/NN
```

Mini Project Faulty Calculator

Design a calculator which will correctly solve all the operations except the following ones: 45*3=555, 56+9=77, 56/6=4. Your program should take one operator and two operand as input from the user and then return the result.

In [55]:
```
operand1=int(input("Enter the first operand: "))
operand2=int(input("Enter the second operand: "))
print("Calculator Function")
print("Addition +")
print("Subtraction -")
print("Multiplication *")
print("division /")
operator=input("Enter the operator: ")
print()
if(operand1==45 and operand2==3 and operator=="*"):
    print("The {} {} {} output is 555 ".format(operand1,operator,operand
elif(operand1==56 and operand2==9 and operator=="+"):
    print("The {} {} {} output is 77".format(operand1,operator,operand2)
elif(operand1==56 and operand2==6 and operator=="/"):
    print("The {} {} {} output is 4".format(operand1,operator,operand2))
elif(operator=="+"):
    print("The {} {} {} output is {} ".format(operand1,operator,operand2
elif(operator=="-"):
    print("The {} {} {} output is {} ".format(operand1,operator,operand2
elif(operator=="*"):
    print("The {} {} {} output is {} ".format(operand1,operator,operand2
elif(operator=="/"):
    print("The {} {} {} output is {} ".format(operand1,operator,operand2
```

```
Enter the first operand: 45
Enter the second operand: 3
Calculator Function
Addition +
Subtraction -
Multiplication *
division /
Enter the operator: *

The 45 * 3 output is 555
```

Nested if statements

A nested if is an if statement that is present in the code block of another if statement. In other words, it means- an if statement inside another if statement. Yes, Python allows such a framework for us to nest if statements. Just like nested if statements, we can have all types of nested conditionals. A nested conditional will be executed only when the parent conditional is true.

Syntax of nested if

In [ ]:
```
if(test expression):
    statement(s)
    if(test expression):
        statement(s)
```

Example

In [56]:
```
num = float(input("Enter a number: "))

if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")

else:
    print("Negative number")
```

```
Enter a number: 25
Positive number
```

Explore via coding:

Write a python program to print if number is even or divisible of 5.

In [57]:
```
num=int(input("Enter the number: "))

if(num%2==0):
    if(num%5==0):
        print("{} is even and divisible by 5 too".format(num))
else:
    print("It is not even or divisible by 5")
```

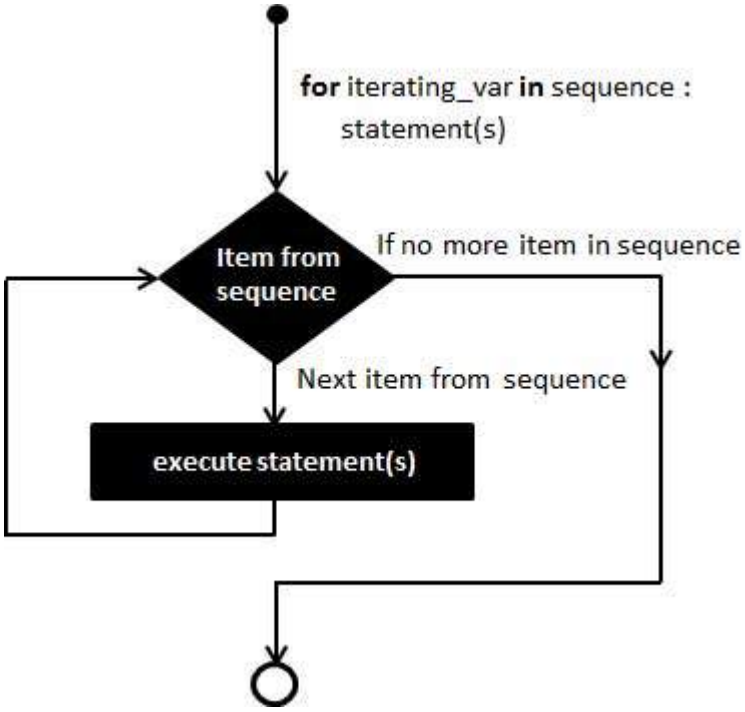
```
Enter the number: 10
10 is even and divisible by 5 too
```

==========

Loops

for Loop

The for loop in Python is used to iterate over a sequence (list, tuple, string,dictionary) or other iterable objects. Iterating over a sequence is called traversal.



Syntax of for Loop

In [ ]:
```
for val in sequence:
    Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Example: Python for Loop

In [58]:
```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

sum = 0

for val in numbers:
    sum = sum+val

print("The sum is", sum)
```

```
The sum is 48
```

Explore via coding:

Write a python program to print table of any particular number.

In [59]:
```
num=int(input("Enter the number you want table: "))

for i in range(1,11):
    print("{} * {} = {}".format(num,i,num*i))
```

```
Enter the number you want table: 8
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
```

```
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80
```

The range() function

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).
We can also define the start, stop and step size as range(start, stop,step_size). step_size defaults to 1 if not provided.
The range object is "lazy" in a sense because it doesn't generate every number that it "contains" when we create it.
However, it is not an iterator since it supports in, len and get item operations.
This function does not store all the values in memory; it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.
To force this function to output all the items, we can use the function list().

Syntax of for Loop

In [ ]:
```
range(Start,End,Step)

#by default the value of Start = 0, End = N, Step = 1
```

Example:

In [60]:
```
print(range(10))

print(list(range(10)))

print(list(range(2, 8)))

print(list(range(2, 20, 3)))
```

```
range(0, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[2, 3, 4, 5, 6, 7]
[2, 5, 8, 11, 14, 17]
```

We can use the range() function in for loops to iterate through a sequence of numbers. It can be combined with the len() function to iterate through a sequence using indexing. Here is an example.

In [61]:
```
genre = ['pop', 'rock', 'jazz']

for i in range(len(genre)):
    print("I like", genre[i])
```

```
I like pop
I like rock
I like jazz
```

for loop with else

A for loop can have an optional else block as well. The else part is executed if the items in the sequence used in for loop exhausts.

The break keyword can be used to stop a for loop. In such cases, the else part is ignored.

Hence, a for loop's else part runs if no break occurs.

Here is an example to illustrate this.

Example:

In [62]:
```
digits = [0, 1, 5]

for i in digits:
    print(i)
else:
    print("No items left.")
```

```
0
1
5
No items left.
```

Code Explanation:

Here, the for loop prints items of the list until the loop exhausts. When the for loop exhausts, it executes the block of code in the else and prints No items left.

This for...else statement can be used with the break keyword to run the else block only when the break keyword was not executed. Let's take an example:

Example

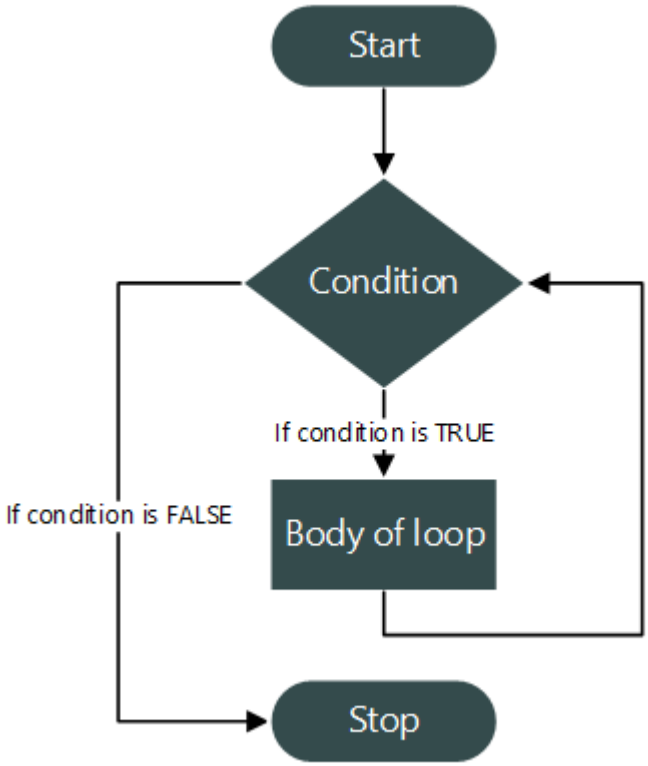
In [63]:
```
student_name = 'Soyuj'

marks = {'James': 90, 'Jules': 55, 'Arthur': 77}

for student in marks:
    if student == student_name:
        print(marks[student])
        break
    else:
        print('No entry with that name found.')
```

```
No entry with that name found.
No entry with that name found.
No entry with that name found.
```

while Loop

The while loop is somewhat similar to an if statement, it executes the code block inside if the expression/condition is True . However, as opposed to the if statement, the while loop continues to execute the code repeatedly, as long as the expression is True . In other words, a while loop iterates over a block of code. In Python, the body of a while loop is determined by the indentation. It starts with indentation and ends at the first unindented line. The most important part of a while loop is the looping variable. This looping variable controls the flow of iterations. An increment or decrement in this looping variable is important for the loop to function. It determines the next iteration level. In case of the

absence of such increment/decrement, the loop gets stuck at the current iteration and continues forever until the process is manually terminated.



Syntax of while Loop in Python

In [ ]:
```
while test_expression:
    Body of while
```

In the while loop, test expression is checked first. The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False.

In Python, the body of the while loop is determined through indentation.

The body starts with indentation and the first unindented line marks the end.

Python interprets any non-zero value as True. None and 0 are interpreted as False.

Example:

In [64]:
```
n = 10

# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1 # update counter

#print the sum
print("The sum is", sum)
```

```
The sum is 55
```

Code Explanation:

In the above program, the test expression will be True as long as our counter variable i is less than or equal to n (10 in our program).

We need to increase the value of the counter variable in the body of the loop. This is very important (and mostly forgotten). Failing to do so will result in an infinite loop (never-ending loop).

Finally, the result is displayed

Explore via coding:

Write a python program to check wheather a number is prime or not.

In [65]:
```
num=int(input("Enter the number you want to check for prime: "))

Flag=True
i=2
while(i<num):
    if(num%i==0):
        Flag=False
        break
    i=i+1

if(Flag):
    print("Number is prime")
else:
    print("Number is not prime")
```

```
Enter the number you want to check for prime: 8
Number is not prime
```

While loop with else

Same as with for loops, while loops can also have an optional else block.

The else part is executed if the condition in the while loop evaluates to False.

The while loop can be terminated with a break statement. In such cases, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.

Here is an example to illustrate this

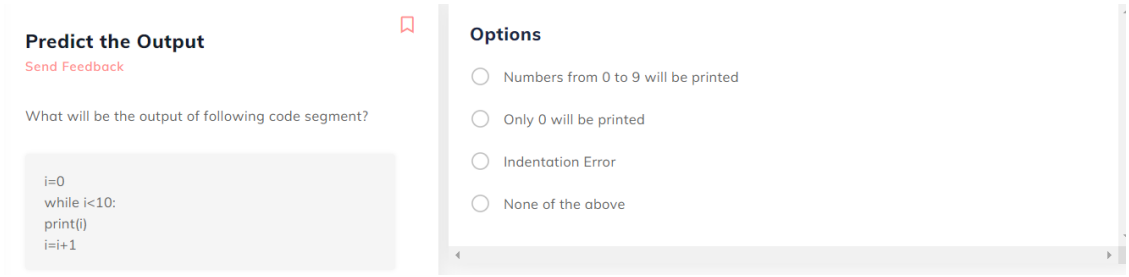
Example:

In [66]:
```
counter = 0

while counter < 3:
    print("Inside loop")
    counter = counter + 1
else:
    print("Inside else")
```

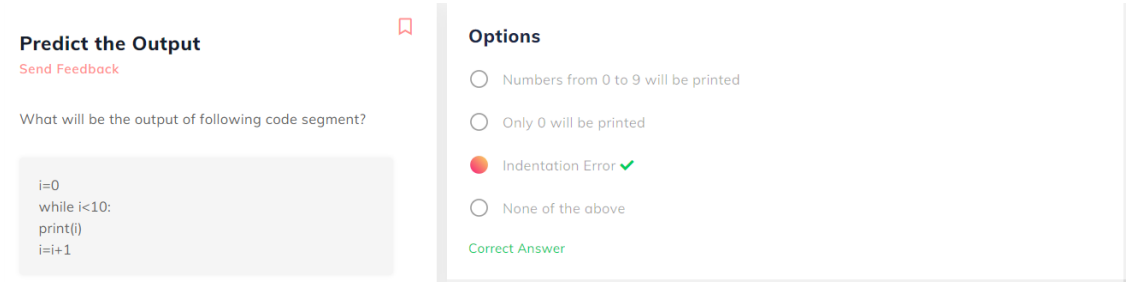
```
Inside loop
Inside loop
```

```
Inside loop
Inside else
```

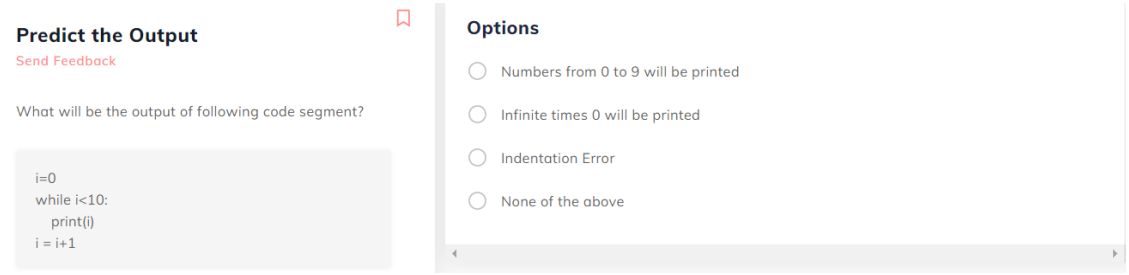
Quiz



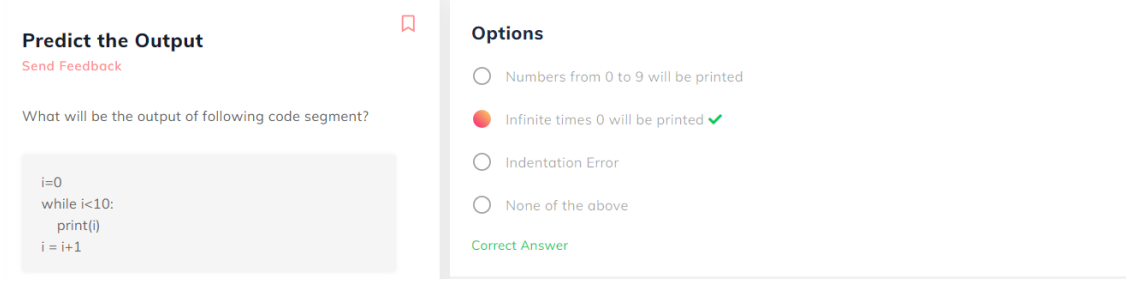
Solution



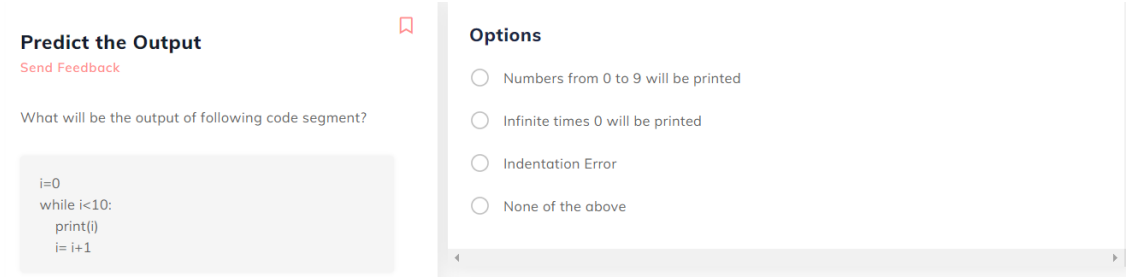
Quiz



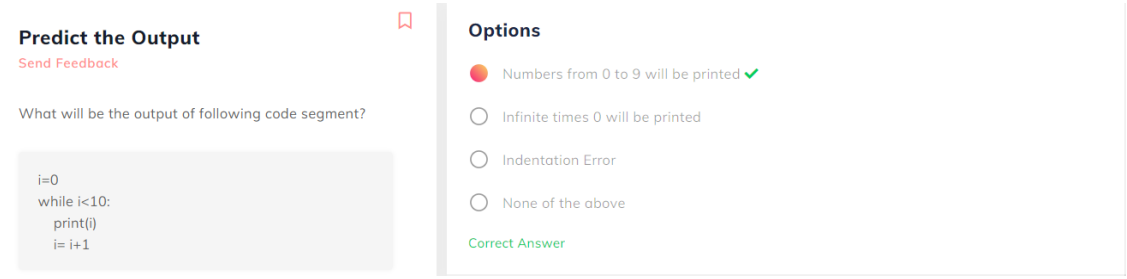
Solution



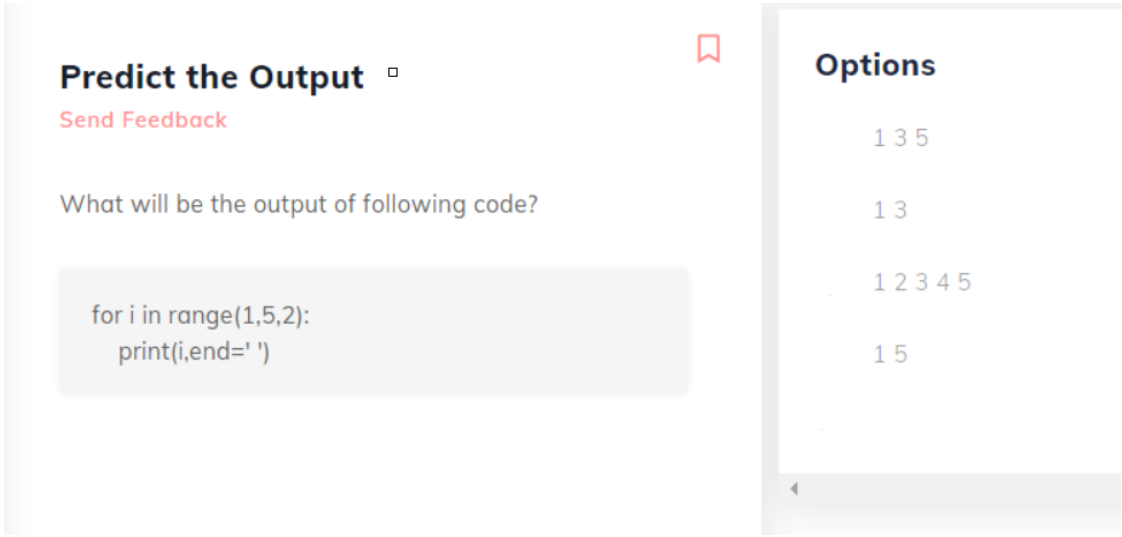
Quiz



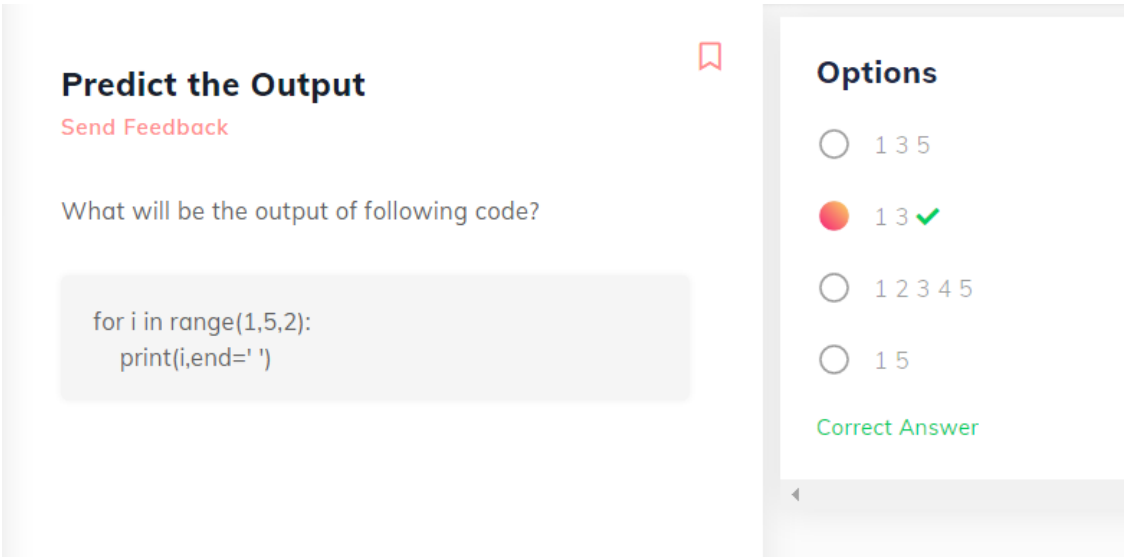
Solution



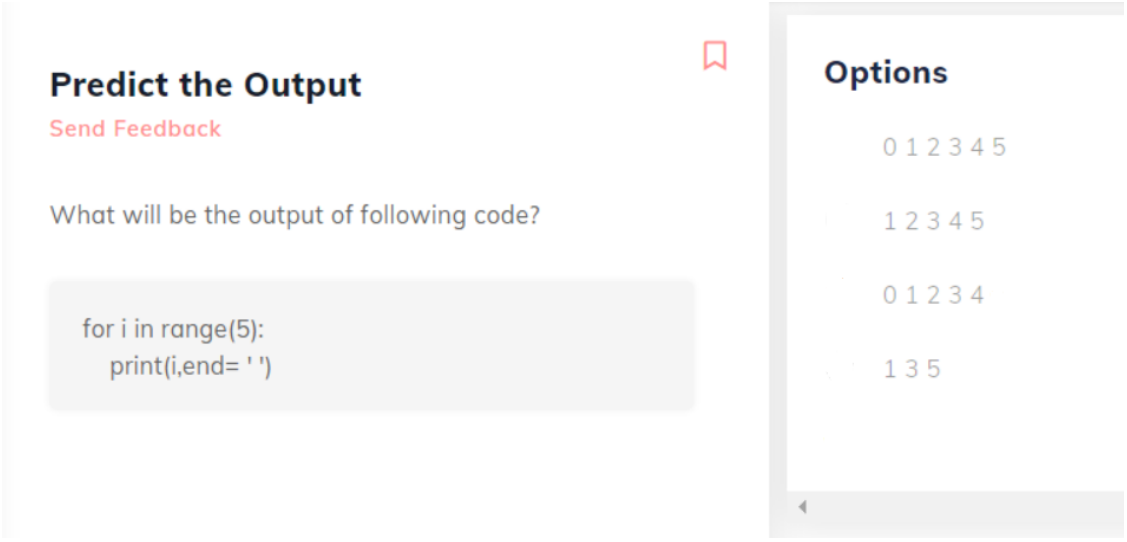
Quiz



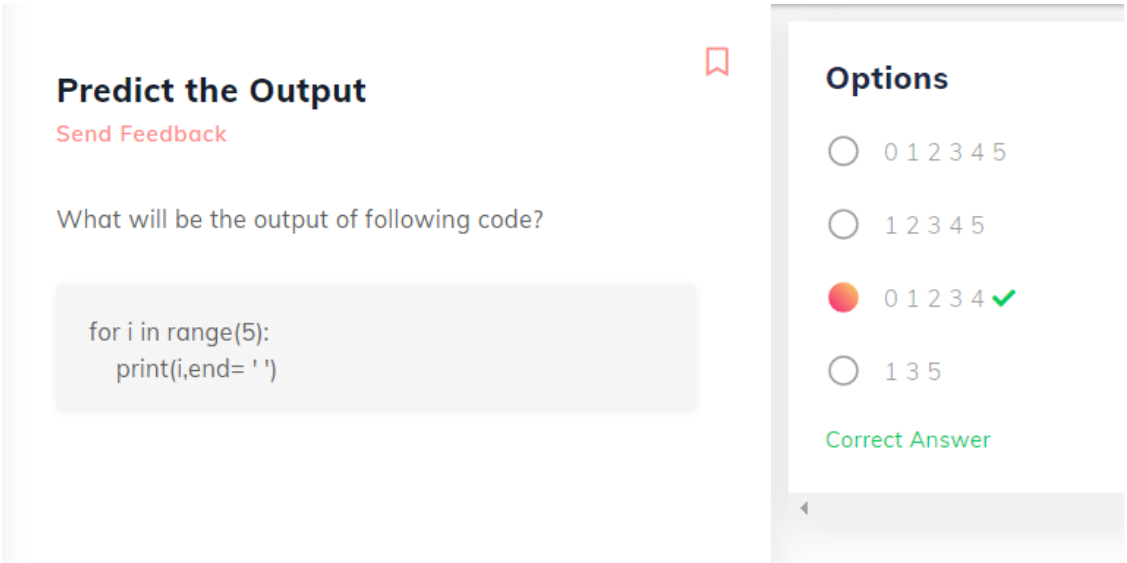
Solution



Quiz



Solution



Quiz

Predict the Output

Send Feedback

What will be the output of following code?

```
i=1
while i<5:
   if i==3:
      break
   print(i,end=" ")
   i = i +1
```

Options

1 2 3 4

1 2

1 2 3

Infinite Loop

Solution

Predict the Output

Send Feedback

What will be the output of following code?

```
i=1
while i<5:
   if i==3:
      break
   print(i,end=" ")
   i = i +1
```

Options

1 2 3 4

1 2 ✔

1 2 3

Infinite Loop

Correct Answer

Quiz

Predict the Output

Send Feedback

What will be the output of following code?

```
i=1
while i<3:
   j=1
   while j<5:
      if j==3:
         break
      print(j,end=" ")
      j = j + 1
   i = i + 1
```

Options

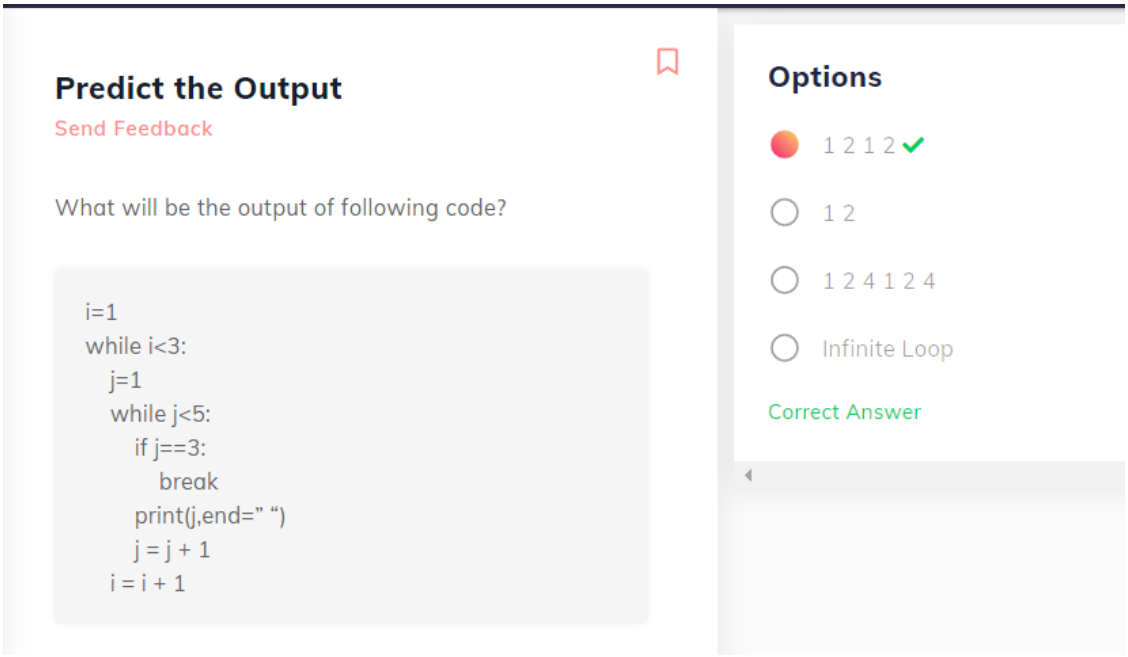
1 2 1 2

1 2

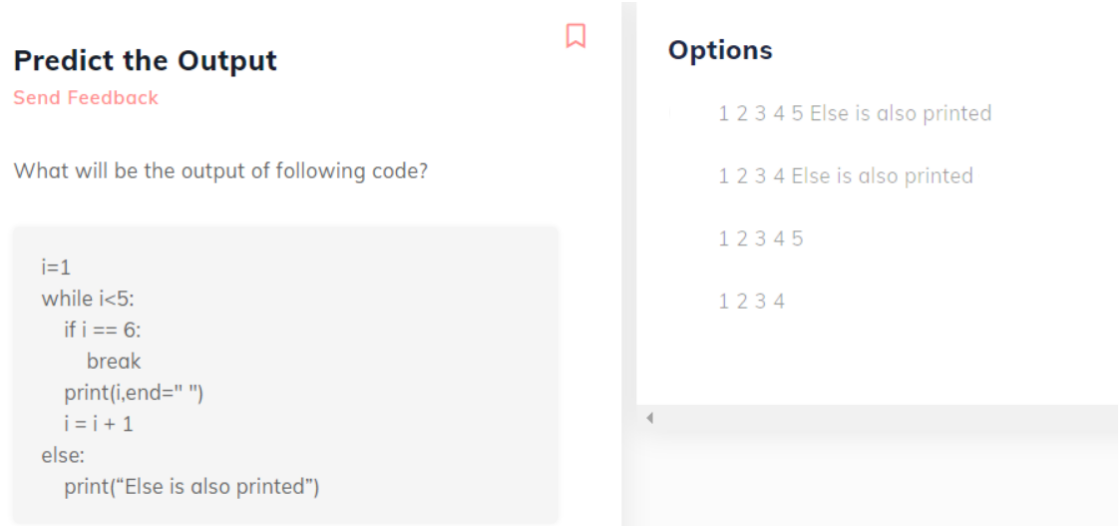
1 2 4 1 2 4

Infinite Loop

Solution



Quiz



Solution

Predict the Output

Send Feedback

What will be the output of following code?

```
i=1
while i<5:
    if i == 6:
        break
    print(i,end=" ")
    i = i + 1
else:
    print("Else is also printed")
```

Options

- 1 2 3 4 5 Else is also printed
- 1 2 3 4 Else is also printed ✓
- 1 2 3 4 5
- 1 2 3 4

Correct Answer

Quiz

Predict the Output

Send Feedback

What will be the output of following code?

```
i=1
while i<5:
    if i == 3:
        break
    print(i,end=" ")
    i = i + 1
else:
    print("Else is also printed")
```

Options

- 1 2 3 4 5 Else is also printed
- 1 2 Else is also printed
- 1 2
- 1 2 3 4

Solution

Predict the Output

Send Feedback

What will be the output of following code?

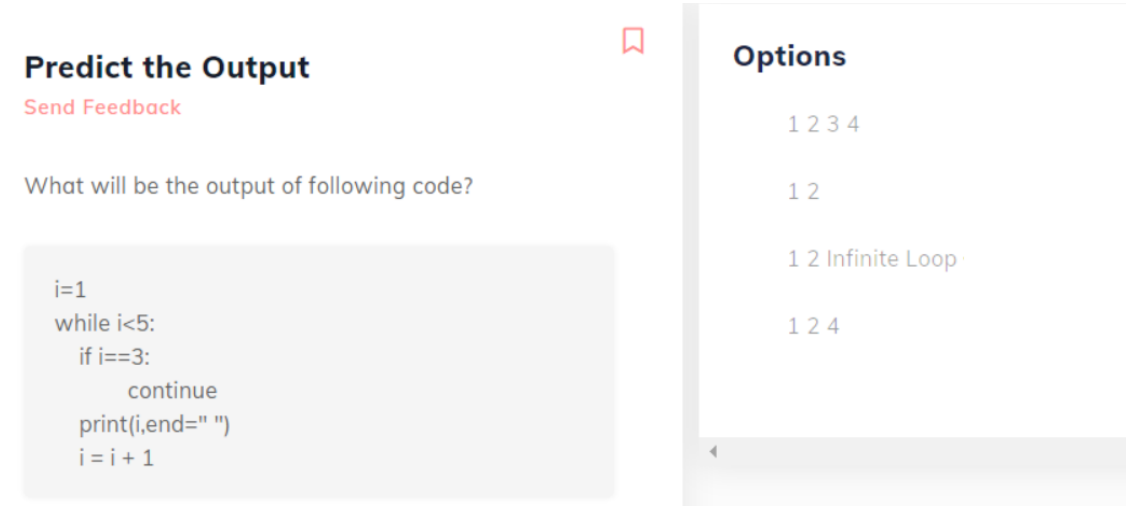
```
i=1
while i<5:
    if i == 3:
        break
    print(i,end=" ")
    i = i + 1
else:
    print("Else is also printed")
```

Options

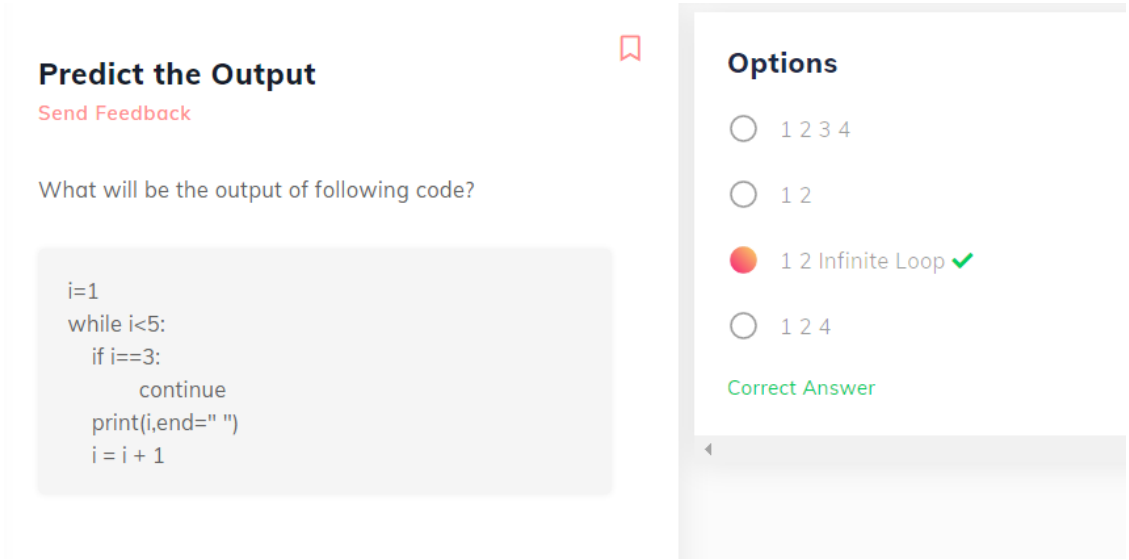
- 1 2 3 4 5 Else is also printed
- 1 2 Else is also printed
- 1 2 ✓
- 1 2 3 4

Correct Answer

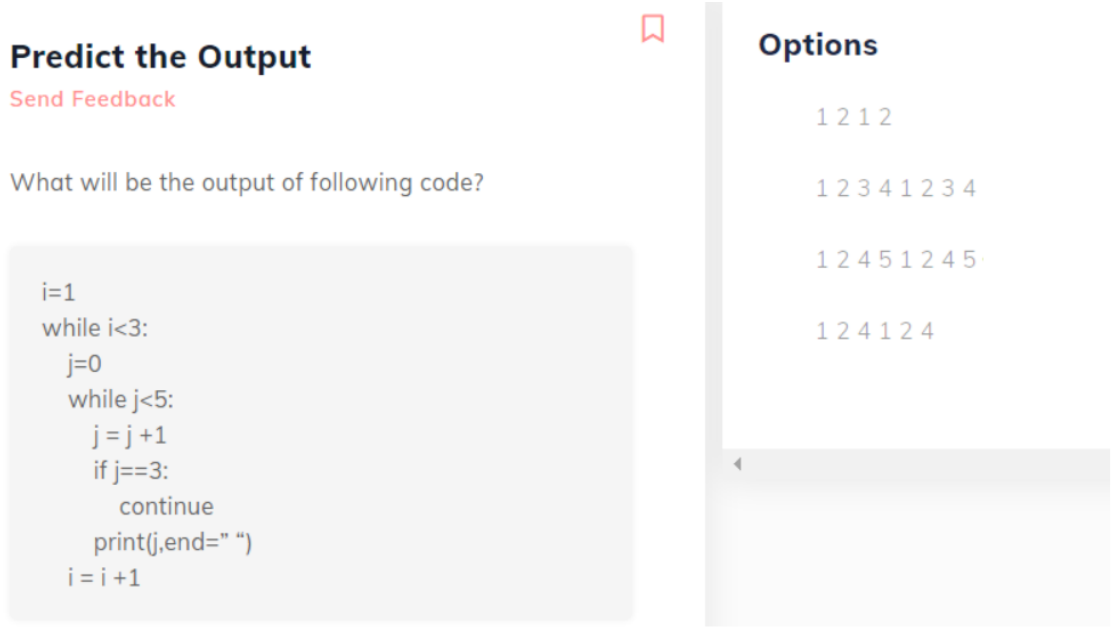
Quiz



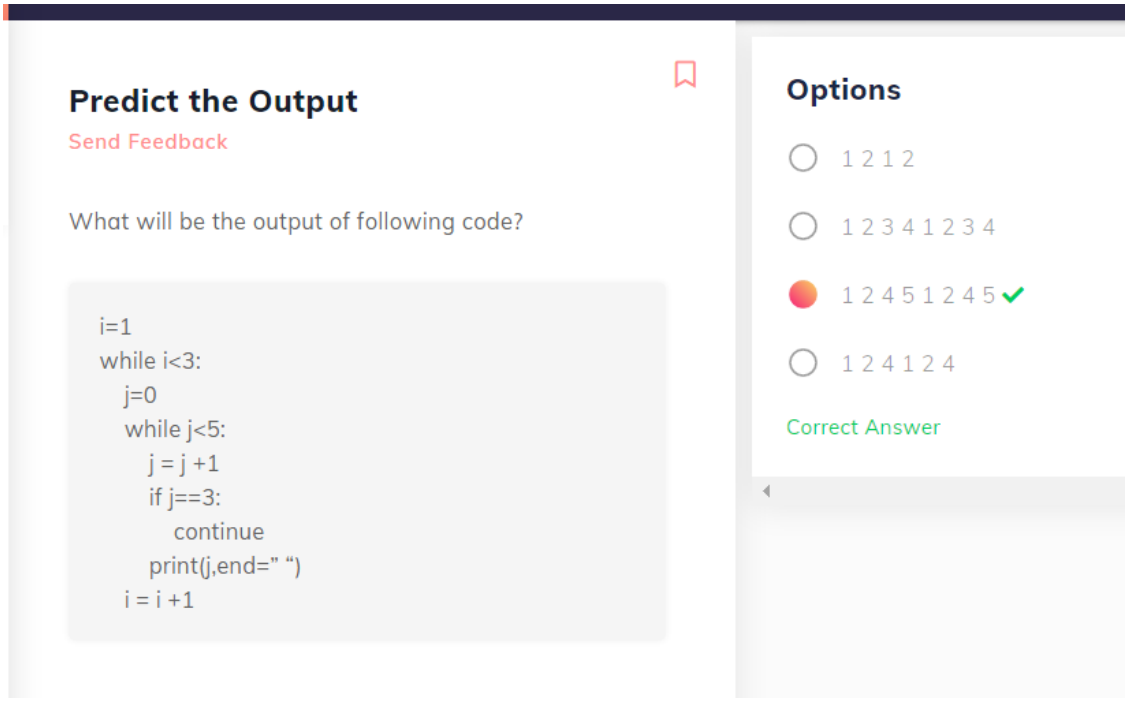
Solution



Quiz



Solution



Explore via coding:

Write a python program to check wheather a number is prime or not.

In [67]:
```
num=int(input("Enter the number you want to check for prime: "))

Flag=True
i=2
while(i<num):
    if(num%i==0):
        print("Number is not prime")
        break
    i=i+1
else:
    print("Number is prime")
```

```
Enter the number you want to check for prime: 13
Number is prime
```

==========

break, continue and pass

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

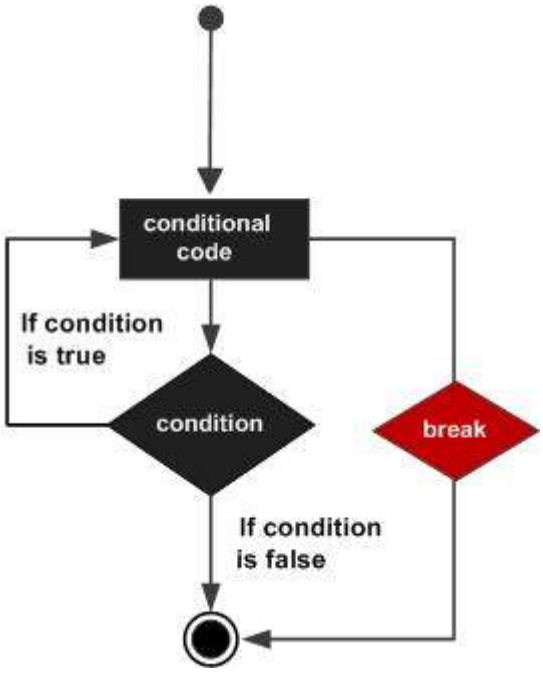
Loops iterate over a block of code until the test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.

The break and continue statements are used in these cases.

break statement

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

If the break statement is inside a nested loop (loop inside another loop), the break statement will terminate the innermost loop.



Example:

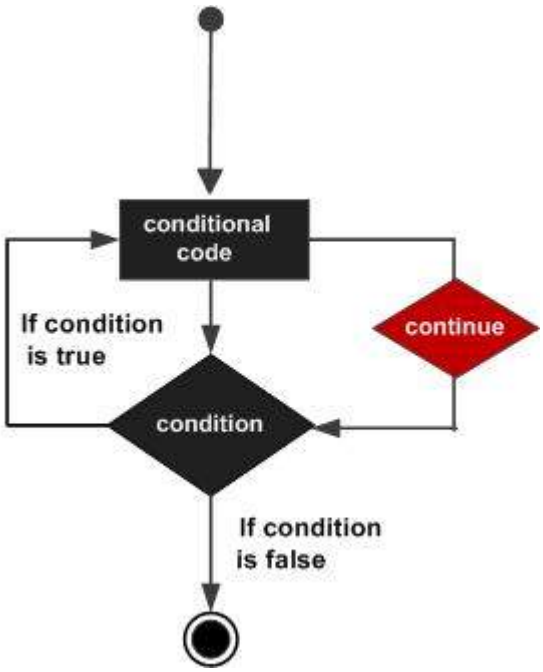
In [68]:
```
for val in "string":
    if val == "i":
        break
    print(val)

print("The end")
```

```
s
t
r
The end
```

Code Explanation: In this program, we iterate through the "string" sequence. We check if the letter is i, upon which we break from the loop. Hence, we see in our output that all the letters up till i gets printed. After that, the loop terminates.

continue statement

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.



Example:

In [69]:
```
for val in "string":
    if val == "i":
        continue
    print(val)

print("The end")
```

```
s
t
r
n
g
The end
```

Code Explanation:

ExplainThis program is same as the above example except the break statement has been replaced with continue.

We continue with the loop, if the string is i, not executing the rest of the block. Hence, we see in our output that all the letters except i gets printed.

In [70]:
```
if True or True:
    if False and True or False:
        print('A')
    elif False and False or True and True:
       print('B')
    else:
```

```
      print('C')
else:
     print('D')
```

```
B
```

Pass Statement

The pass statement is a null statement.

- It is generally used as a placeholder for future code i.e. in cases where you want to implement some part of your code in the future but you cannot leave the space blank as it will give a compilation error.
- Sometimes, pass is used when the user does not want any code to execute.
- Using the pass statement in loops, functions, class definitions, and if statements, is very useful as empty code blocks are not allowed in these.

Syntax:

In [71]:
```
pass
```

Example:

In [72]:
```
n=2
if n== 2 :
    pass #Pass statement: Nothing will happen
else :
    print("Executed")
```

Code explanation: In the above code, the if statement condition is satisfied because n== 2 is True . Thus there will be no output for the above code because once it enters the if statement block, there is a pass statement. Also, no compilation error will be produced.

Example:

In [73]:
```
n= 1
if n== 2 :
    print("Executed")
else :
    pass #Pass statement: Nothing will happen
```

Code explanation: In the above code, the if statement condition is not satisfied because n== 2 is False . Thus there will be no output for the above code because it enters the else statement block and encounters the pass statement.

Mini Project Guess the number

Problem statement is in system we have fixed a number and user have to guess it in 5 chances and each and every guess you have to tell user whatever number you choose is larger or smaller than the fixed number and if he/she succesfully guess the number print "You won" or he/she do not able to guess then print "You loose".

In [74]:
```
fixed_number=20
turn=5

print("Let's start the game buddy")
```

```
while(turn>=1):
    print()
    print("You have {} chance to guess the number".format(turn))
    guess_no=int(input("Enter your guess number: "))
    if(guess_no>fixed_number):
        print("Your guess number is greater")
    elif(guess_no<fixed_number):
        print("Your guess number is smaller")
    elif(guess_no==fixed_number):
        print("Hey! Congratulations you won the game")
        break
    turn=turn-1
    if(turn==0):
        print()
        print("You lose the game better luck next time")
```

```
Let's start the game buddy

You have 5 chance to guess the number
Enter your guess number: 5
Your guess number is smaller

You have 4 chance to guess the number
Enter your guess number: 30
Your guess number is greater

You have 3 chance to guess the number
Enter your guess number: 10
Your guess number is smaller

You have 2 chance to guess the number
Enter your guess number: 25
Your guess number is greater

You have 1 chance to guess the number
Enter your guess number: 20
Hey! Congratulations you won the game
```

==========

Function:

In Python, a function is a group of related statements that performs a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes the code reusable.

Advantages of function:

1. It helps to divide the large programs into small groups so that we can read the code, and debug the program faster and better.
2. Python Functions stop us from writing the same logic various times. We can bind the logic in one function and then call the same over and over.
3. It encourages us to call the same function with different inputs over multiple times.
4. Through function implementation in program, program readibility get increased.
5. The testing of program will become easy, if you will encountered with any error you just have to debug only that function.

Syntax of a Function

In [ ]:
```
def function_name(parameters):

    body of function
```

Defining Functions In Python

A function, once defined, can be invoked as many times as needed by using its name, without having to rewrite its code.

Syntax:

In [ ]:
```
def <function-name>(<parameters>):
    """ Function's docstring """
    <Expressions/Statements/Instructions>
```

Function blocks begin with the keyword def followed by the function name and parentheses (( )).

- The input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function is optional - the documentation string of the function or docstring. The docstring describes the functionality of a function.
- The code block within every function starts with a colon (:) and is indented. All statements within the same code block are at the same indentation level.
- The return statement exits a function, optionally passing back an expression/value to the function caller.

Let us define a function to add two numbers.

In [75]:
```
def add(a,b):
    return a+b
```

The return Statement

A return statement is used to end the execution of the function call and it “returns” the result (value of the expression following the return keyword) to the caller. The statements after the return statements are not executed. If the return statement is without any expression, then the special value None is returned.

In the example given above, the sum a+b is returned.

Note: In Python, you need not specify the return type i.e. the data type of returned value.

Calling/Invoking A Function

Once you have defined a function, you can call it from another function, program, or even the Python prompt. To use a function that has been defined earlier, you need to write a function call.

A function call takes the following form:

In [ ]:
```
<function-name> (<value-to-be-passed-as-argument>)
```

The function definition does not execute the function body. The function gets executed

only when it is called or invoked. To call the above function we can write:

In [76]:
```
add(5,7)
```

Out[76]:
```
12
```

In this function call, a = 5 and b = 7.

Arguments And Parameters

As you know that you can pass values to functions. For this, you define variables to receive values in the function definition and you send values via a function call statement. For example, in the **add()** function, we have variables a and b to receive the values and while calling the function we pass the values 5 and 7. We can define

These two types of values:

- **Arguments:** The values being passed to the function from the function call statement are called arguments. Eg. 5 and 7 are arguments to the add() function.
- **Parameters:** The values received by the function as inputs are called parameters. Eg. a and b are the parameters of the add() function.

Types Of Functions

We can divide functions into the following two types:

1. **User-defined functions:** Functions that are defined by the users. Eg. The add() function we created.
2. **Inbuilt Functions:** Functions that are inbuilt in python. Eg. The print() function.

The Lifetime of a Variable

The lifetime of a variable is the time for which the variable exists in the memory.

- The lifetime of a Global variable is the entire program run (i.e. they live in the memory as long as the program is being executed).
- The lifetime of a Local variable is their function's run (i.e. as long as their function is being executed).

Explore via coding:

Write a python program for calculator using function.

In [77]:
```
def calculator(opr1,opr2,opr):
    if(opr=="+"):
        print("Addition of two number is: ",opr1+opr2)
    elif(opr=="-"):
        print("Subtraction of two number is: ",opr1-opr2)
    elif(opr=="*"):
        print("Multiplication of two number is: ",opr1*opr2)
    elif(opr=="/"):
        print("Division of two number is: ",opr1/opr2)
```

```
opr1=int(input("Enter the first operand: "))
opr2=int(input("Enter the second operand: "))
opr=input("Enter the operator: ")

calculator(opr1,opr2,opr)
```

```
Enter the first operand: 27
Enter the second operand: 3
Enter the operator: *
Multiplication of two number is:  81
```

==========

Types of variables:

All variables in a program may not be accessible at all locations in that program. Part(s) of the program within which the variable name is legal and accessible, is called the scope of the variable. A variable will only be visible to and accessible by the code blocks in its scope.

Local Variable: Variables that are defined inside a function body have a local scope. This means that local variables can be accessed only inside the function in which they are declared.

Local variable can be used by only inside the function.

Example:

In [78]:
```
def foo():                    #function defination
    y = "local"
    print(y)

foo()                         #function calling
```

```
local
```

Accessing A Local Variable Outside The Scope:

In [79]:
```
def foo():
    y = "local"
foo()
print(y)
```

```
5
```

In the above code, we declared a local variable y inside the function foo(), and then we tried to access it from outside the function.

We get the output as **"name 'y' is not defined"**

We get an error because the lifetime of a local variable is the function it is defined in. Outside the function, the variable does not exist and cannot be accessed. In other words, a variable cannot be accessed outside its scope.

Global variable: A variable/name declared in the top-level segment (__main__) of a program is said to have a global scope and is usable inside the whole program (Can be accessed from anywhere in the program). In Python, a variable declared outside a

function is known as a global variable. This means that a global variable can be accessed from inside or outside of the function.

Global variables can be used by everyone, both inside of function and outside of function.

Example:

In [80]:
```
x="Awesome"
def myfunc():                    #function definition
    print("Python is: "+x)

myfunc()                         #function calling
```

```
Python is: Awesome
```

Here, we created a global variable x = "Awesome". Then, we created a function myfunc() to print the value of the global variable from inside the function. We get the output as "Awesome".

Thus we can conclude that we can access a global variable from inside any function.

NOTE: If you create a variable with the same name inside a function, this variable will be local,and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

What if you want to change the value of a Global Variable from inside a function?

Consider the code snippet:

In [81]:
```
x = "Global Variable"
def foo():
    x = "Awesome"
    print(x)
foo()
print(x)
```

```
Awesome
Global Variable
```

Code Explanation: As you can see from above code snippet variable "x" which is outside the foo() function is global variable and if you create the variable with same name and print in the scope of function then the variable "x" which is inside the foo treated as a local variable and local variable has more priority then global variable.

The global keyword: Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.
To create a global keyword inside a function, you can use global keyword

Example:If you use the global keyword, the variable belongs to the global scope

In [82]:
```
def myfunc():                    #function definition
    global x
    x="Fantastic"

myfunc()                         #function call
print("Python is: "+x)
```

```
Python is: Fantastic
```

Global Variable And Local Variable With The Same Name

Consider the code given:

In [83]:
```
x = 5
def foo():
    x = 10
    print("Local:", x)
foo()
print("Global:", x)
```

```
Local: 10
Global: 5
```

In this, we have declared a global variable x = 5 outside the function foo(). Now, inside the function foo(), we re-declared a local variable with the same name, x. Now, we try to print the values of x, inside, and outside the function. We observe the following output:

Local: 10
Global: 5

In the above code, we used the same name x for both global and local variables. We get a different result when we print the value of x because the variables have been declared in different scopes, i.e. the local scope inside foo() and global scope outside foo().

When we print the value of the variable inside foo() it outputs Local: 10. This is called the local scope of the variable. In the local scope, it prints the value that it has been assigned inside the function.

Similarly, when we print the variable outside foo(), it outputs global Global: 5. This is called the global scope of the variable and the value of the global variable x is printed.

==========

Function with default parameters:

Now we are familier with the function in python. How we can create function. How we can call function. How function works. As you can see in function declaration time we have to pass the arguments. we can pass no arguments we can pass 1 we can pass more than one. And at the time of calling function you have to pass the exact arguments if you fail to pass exact arguments than compiler will show you error.

Now we are going to learn a very interesting concpet in python known as function with default parameters. Let us assume if you have a function whose work is to sum the variable. If you want that at the time of calling you pass 2 variable you get the result of sum of two variable if you pass three variable you will get the sum of three variable like that. Is it possible. Let us have a look:

Example:

In [84]:
```
def sum(a,b,c=0,d=0):
    return a+b+c+d

result1=sum(2,3)
print("The sum of two variable: ",result1)

result2=sum(2,3,5)
print("The sum of two variable: ",result2)
```

```
result3=sum(2,3,5,8)
print("The sum of two variable: ",result3)
```

```
The sum of two variable:  5
The sum of two variable:  10
The sum of two variable:  18
```

How is it possisble???????

Let us have a look to understand this concept of function with default variable. As you can see i have passed 4 variable in argument list. Notice that last two variable i assigned with 0("Zero"). So this sum function work like this. If you pass two variable it will take value of a and b as you passed in function calling argument list and if you pass three variable it will take value of a,b and c as you passed in function calling argument list, and if you pass four variable if will take value of a,b,c and d as you passed at the time of function calling argument list.

But here is one more catch if you pass more than four variable it will throw you error. Let us see with code

In [85]:
```
def sum(a,b,c=0,d=0):
    return a+b+c+d

result1=sum(2,3)
print("The sum of two variable: ",result1)

result2=sum(2,3,5)
print("The sum of two variable: ",result2)

result3=sum(2,3,5,8)
print("The sum of two variable: ",result3)

result4=sum(2,3,5,8,5)
print("The sum of five variable: ",result4)
```

```
The sum of two variable:  5
The sum of two variable:  10
The sum of two variable:  18
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-85-dddfbf0bdd41> in <module>
     11 print("The sum of two variable: ",result3)
     12
---> 13 result4=sum(2,3,5,8,5)
     14 print("The sum of five variable: ",result4)

TypeError: sum() takes from 2 to 4 positional arguments but 5 were given
```

As you can see it is saying you can pass 2 to 4 positional argument not more than four variable.

Let us do one more intresting thing

As you can see If we pass a and b it take c and d as 0("Zero") and if we pass a, b and c it takesit takes d as 0("Zero"). can we do one thing that we pass value of a and c and it takes the default value of b and d. Let us see:

In [86]:
```
def sum(a,b=0,c,d=0):
    return a+b+c+d
```

```
result1=sum(2,3)
print("The sum of two variable: ",result1)
```

```
  File "<ipython-input-86-a2d26e20b945>", line 1
    def sum(a,b=0,c,d=0):
            ^
SyntaxError: non-default argument follows default argument
```

OOps we can not do this we have to make sure that at the time of function declaration parameter list all the non default argument should come in left side and all default should come right side.

Explore via coding:

Write a python program if you give two number then it will show addition of two number if you give three it will give you addition of three number build a program upto five number.

In [87]:
```
def add(num1=0,num2=0,num3=0,num4=0,num5=0):
    return num1+num2+num3+num4+num5

print(add(10))
print(add(10,20))
print(add(10,20,30))
print(add(10,20,30,40))
print(add(10,20,30,40,50))
```

```
10
30
60
100
150
```

==========

Anonymous/Lambda function:

In Python, an anonymous function is a function that is defined without a name.

While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword.

Hence, anonymous functions are also called lambda functions.

Syntax of Lambda Function

In [ ]:
```
lambda arguments: expression
```

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned . Lambda functions can be used wherever function objects are required.

Example

In [88]:
```
double = lambda x: x * 2

print(double(5))
```

```
10
```

Code Explain

In the above program, lambda x: x *2 is the lambda function. Here x is the argument and x* 2 is the expression that gets evaluated and returned.

This function has no name. It returns a function object which is assigned to the identifier double. We can now call it as a normal function. The statement

==========

Passing list as a arguments

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

Example

In [89]:
```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

```
apple
banana
cherry
```

==========

Python *args

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly:

Example

In [90]:
```
def my_function(*food):
  print("The favourite food is " + food[2])
  print(type(food))

my_function("apple", "mango", "banana")
```

```
The favourite food is banana
<class 'tuple'>
```

==========

Recusrion

Introduction

The process in which a function calls itself is called recursion and the corresponding function is called a **recursive function**.

Since computer programming is a fundamental application of mathematics, so let us first try to understand the mathematical reasoning behind recursion.
In general, we all are aware of the concept of functions. In a nutshell, functions are mathematical equations that produce an output on providing input.
For example: Suppose the function **F(x)** is a function defined by:

$$F(x) = x^2 + 4$$

We can write the **Python Code** for this function as:

In [91]:
```
def F(x):
    return(x * x + 4)
```

Now, we can pass different values of x to this function and receive our output accordingly.
Before moving onto the recursion, let's try to understand another mathematical concept known as the Principle of Mathematical Induction (PMI).
Principle of Mathematical Induction (PMI) is a technique for proving a statement, a formula, or a theorem that is asserted about a set of natural numbers. It has the following three steps:

1. **Step of the trivial case:** In this step, we will prove the desired statement for a base case like **n = 0** or **n = 1**.
2. **Step of assumption:** In this step, we will assume that the desired statement is valid for **n = k**.
3. **To prove step:** From the results of the assumption step, we will prove that,**n = k + 1** is also true for the desired equation whenever **n = k** is true.

For Example: Let's prove using the Principle of Mathematical Induction that:

$$S(N): 1 + 2 + 3 + ... + N = (N * (N + 1))/2$$

(The sum of first N natural numbers)
Proof:
Step 1: For N = 1, S(1) = 1 is true.
Step 2: Assume, the given statement is true for N = k, i.e.,
1 + 2 + 3 + ... + k = (k *(k + 1))/2*
***Step 3:** Let's prove the statement for N = k + 1 using step 2.*
To Prove: 1 + 2 + 3 + ... + (k+1) = ((k+1) (k+2))/2
Proof:
Adding (k+1) to both LHS and RHS in the result obtained on step 2:
1 + 2 + 3 + ... + (k+1) = (k*(k+1))/2 + (k+1)*
Now, taking (k+1) common from RHS side:
1 + 2 + 3 + ... + (k+1) = (k+1)((k + 2)/2) According the statement that we are trying to prove:
1 + 2 + 3 + ... + (k+1) = ((k+1)*(k+2))/2
Hence proved.

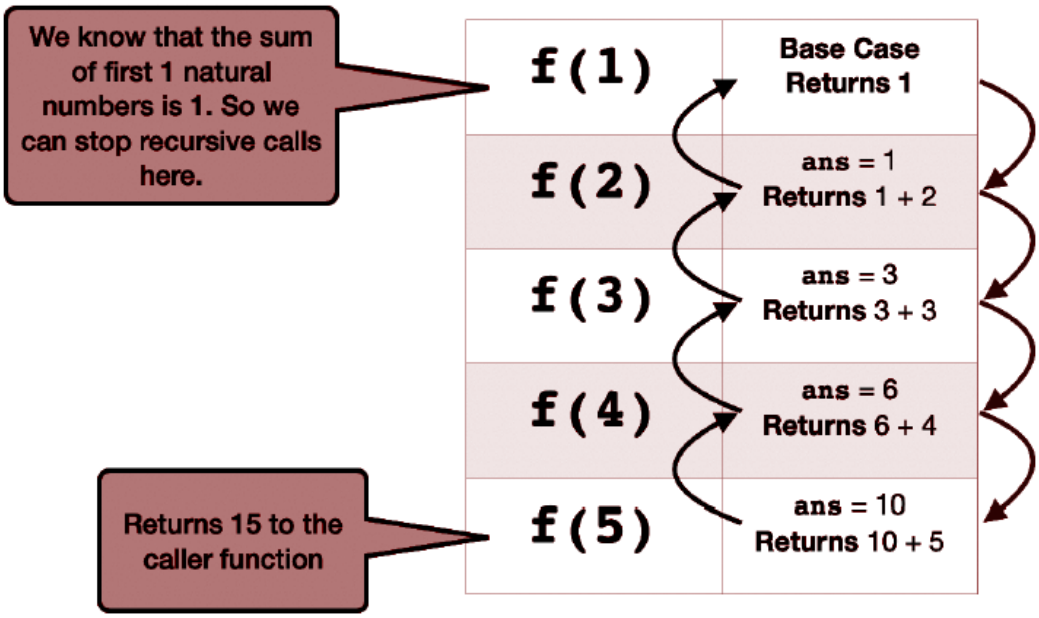
One can think, why are we discussing these over here. To answer this question, we need to know that these three steps of PMI are related to the three steps of recursion, which are as follows:

1. **Induction Step and Induction Hypothesis:** Here, the Induction Step is the main problem which we are trying to solve using recursion, whereas the Induction Hypothesis is the sub-problem, using which we'll solve the induction step. Let's define the Induction Step and Induction Hypothesis for our running example:
   Induction Step: Sum of first n natural numbers - F(n).
   Induction Hypothesis: This gives us the sum of the first n-1 natural numbers - F(n-1)

2. Express F(n) in terms of F(n-1) and write code:
   F(N) = F(N-1)+ N
   Thus, we can write the Python code as:

In [92]:
```
def f(N):
    ans = f(N-1) #Induction Hypothesis step
    return ans + N #Solving problem from result in previous step
```

1. The code is still not complete. The missing part is the base case. Now we will dry run to find the case where the recursion needs to stop.



1. After the dry run, we can conclude that for N equals 1, the answer is 1, which we already know. So we'll use this as our base case. Hence the final code becomes:

In [93]:
```
def f(N):
    if(N == 1): #Base Case
        return 1
    ans = f(N-1)
    return ans + N
```

This is the main idea to solve recursive problems. To summarize, we will always focus on finding the solution to our starting problem and tell the function to compute the rest for us using the particular hypothesis. This idea will be studied in detail in further sections with more examples.

Now, we'll learn more about recursion by solving problems which contain smaller subproblems of the same kind. Recursion in computer science is a method where the solution to the question depends on solutions to smaller instances of the same problem. By the exact nature, it means that the approach that we use to solve the original problem can be used to solve smaller problems as well. So, in other words, in recursion, a function calls itself to solve smaller problems. Recursion is a popular approach for solving problems because recursive solutions are generally easier to think than their iterative counterparts, and the code is also shorter and easier to understand.

Working of recursion

We can define the steps of the recursive approach by summarizing the above three steps:

- **Base case:** A recursive function must have a terminating condition at which the process will stop calling itself. Such a case is known as the base case. In the absence of a base case, it will keep calling itself and get stuck in an infinite loop. Soon, the recursion depth* will be exceeded and it will throw an error.
- **Recursive call:** The recursive function will invoke itself on a smaller version of the main problem. We need to be careful while writing this step as it is crucial to correctly figure out what your smaller problem is.
- **Small calculation:** Generally, we perform a calculation step in each recursive call. We can achieve this calculation step before or after the recursive call depending upon the nature of the problem.

 Note: Recursion uses an in-built stack which stores recursive calls. Hence, the number of recursive calls must be as small as possible to avoid memory-overflow. If the number of recursion calls exceeded the maximum permissible amount, the recursion depth* will be exceeded.

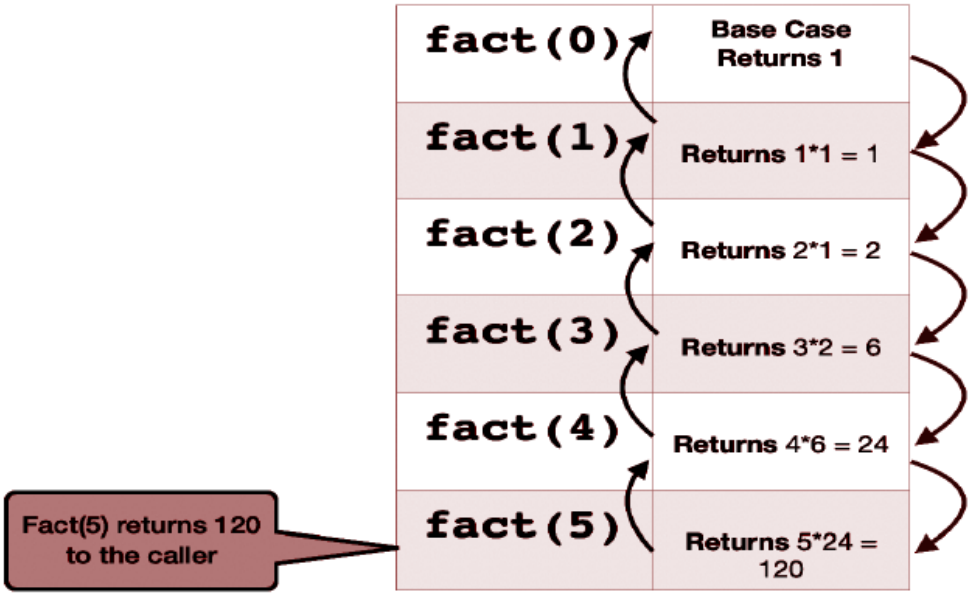
Problem Statement - Find Factorial of a Number

Approach: Figuring out the three steps of PMI and then relating the same using recursion.

1. **Induction Step:** Calculating the factorial of a number n - F(n).
   Induction Hypothesis: We have already obtained the factorial of n-1 - F(n-1).
2. Expressing F(n) in terms of F(n-1): F(n)=n*F(n-1). Thus we get:

In [94]:
```
def fact(n):
    ans = fact(n-1)  #Assumption step
    return ans * n;  #Solving problem from assumption step
```

1. The code is still not complete. The missing part is the base case. Now we will dry run to find the case where the recursion needs to stop. Consider **n = 5**:



As we can see above, we already know the answer of n = 0, which is 1. So we will keep this as our base case. Hence, the code now becomes:

In [96]:
```
def factorial(n):
    if n == 0: #base case
        return 1
    else:
        return n*factorial(n-1) # recursive case
factorial(10)
```

Out[96]: 3628800

Quiz

Predict the Output

Send Feedback

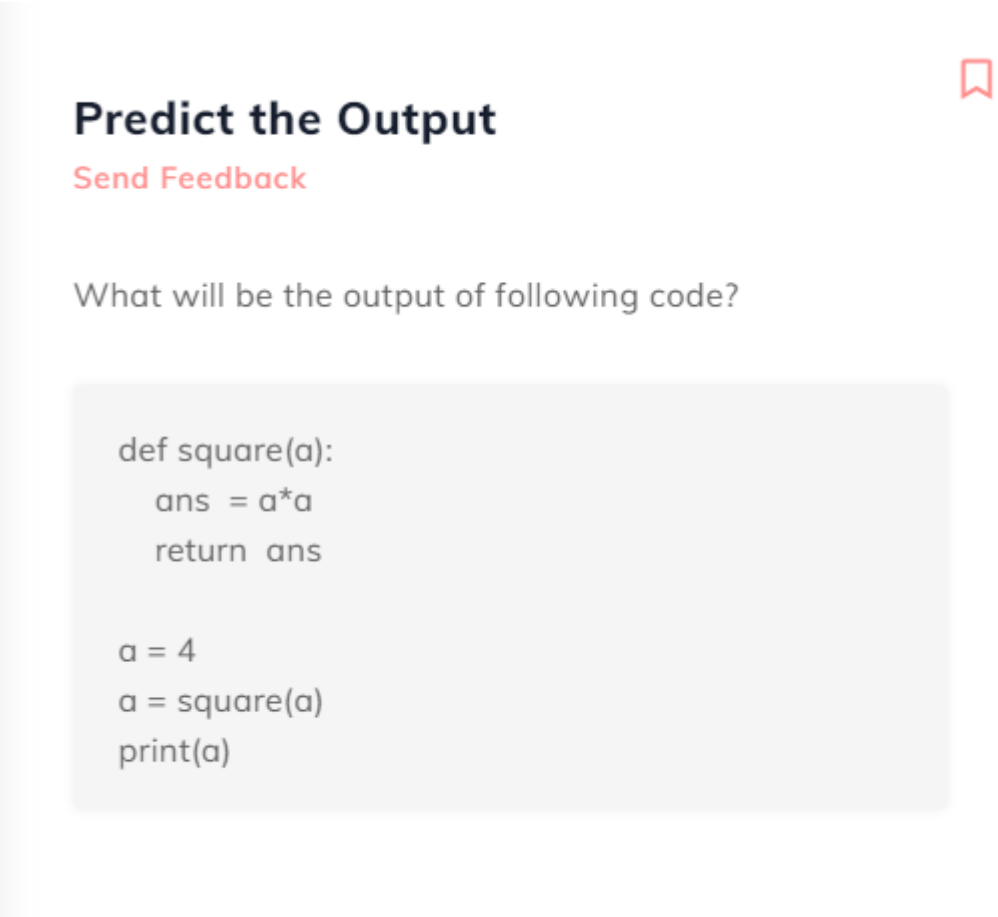
What will be the output of following code?

```
def func(a):
    a = a + 10
    return a
a = 5
func(a)
print(a)
```

Solution

Output of the following code is 5

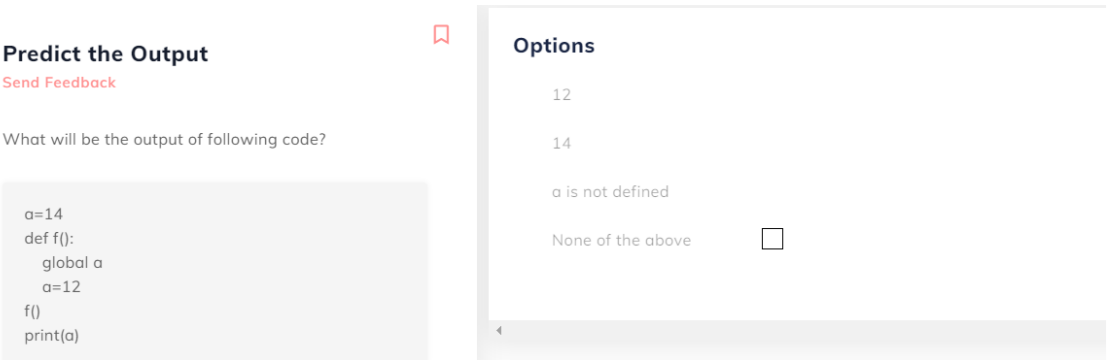
Quiz



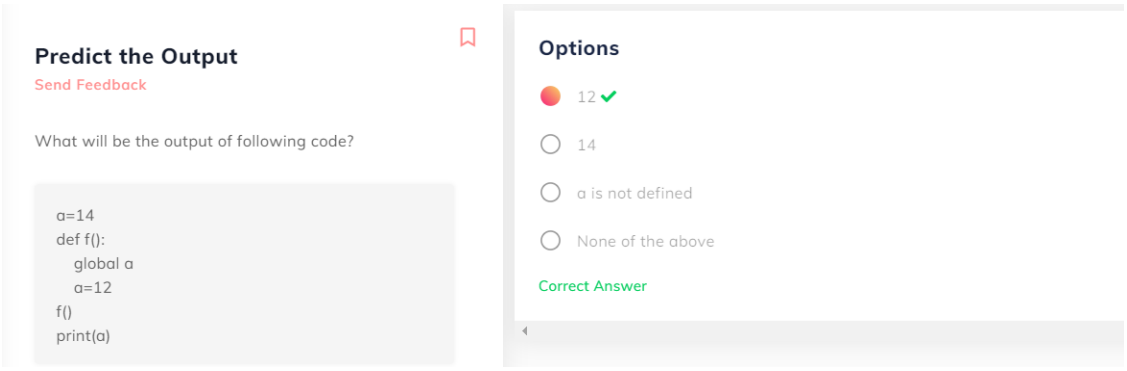
Solution

Output of the following code is 16

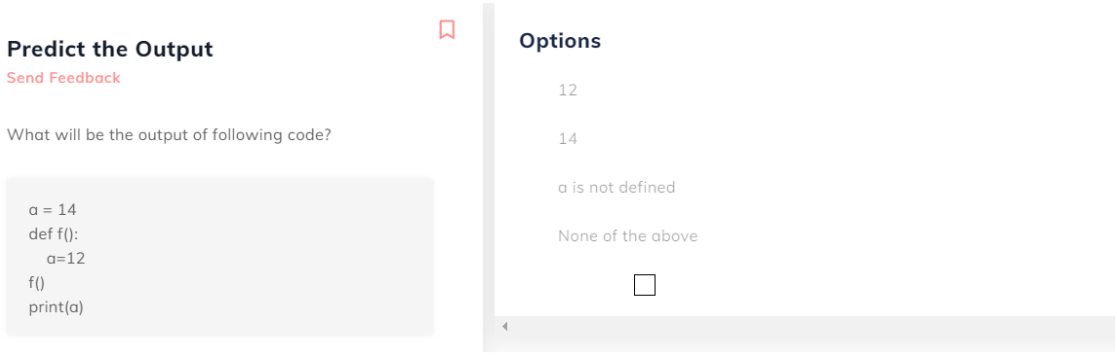
Quiz



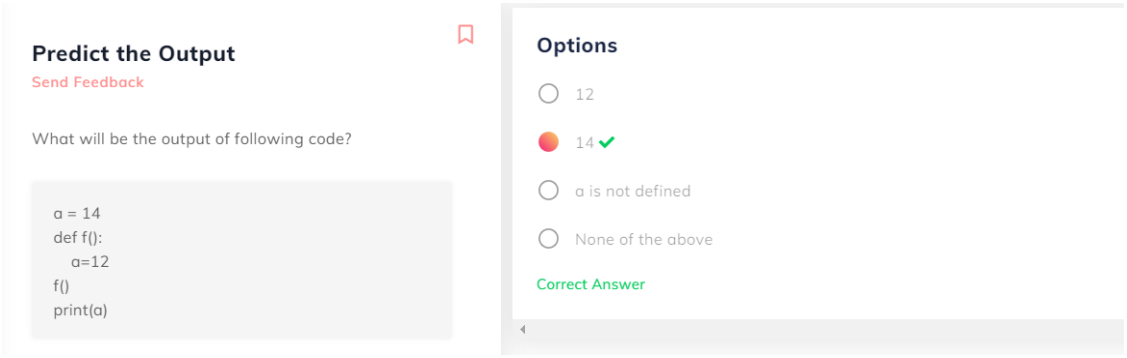
Solution



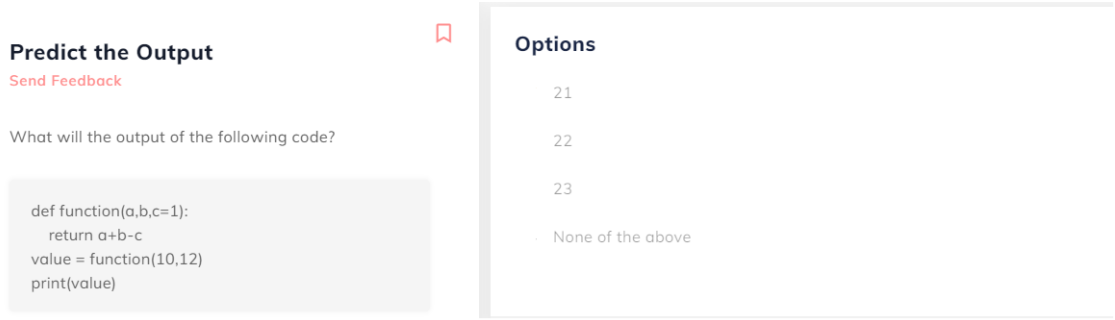
Quiz



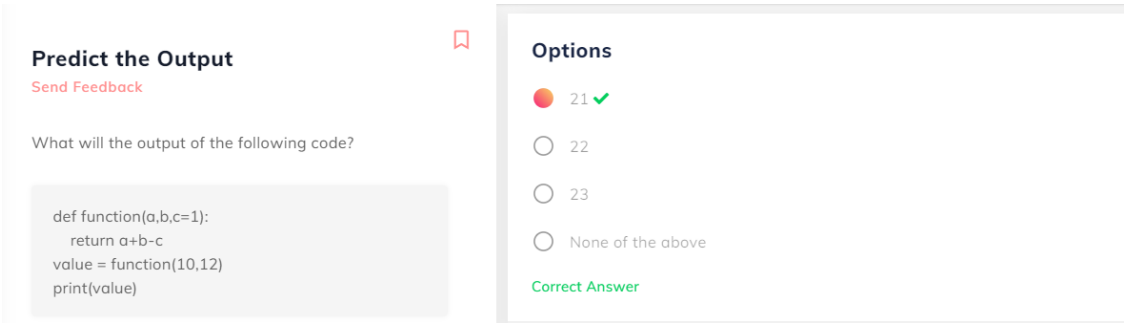
Solution



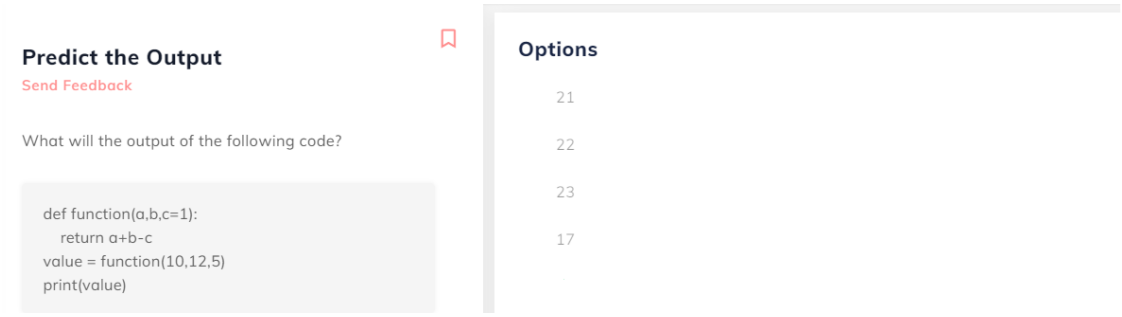
Quiz



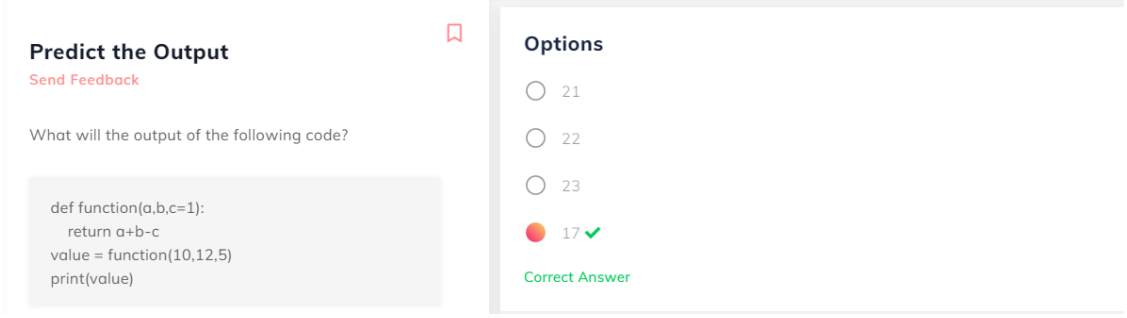
Solution



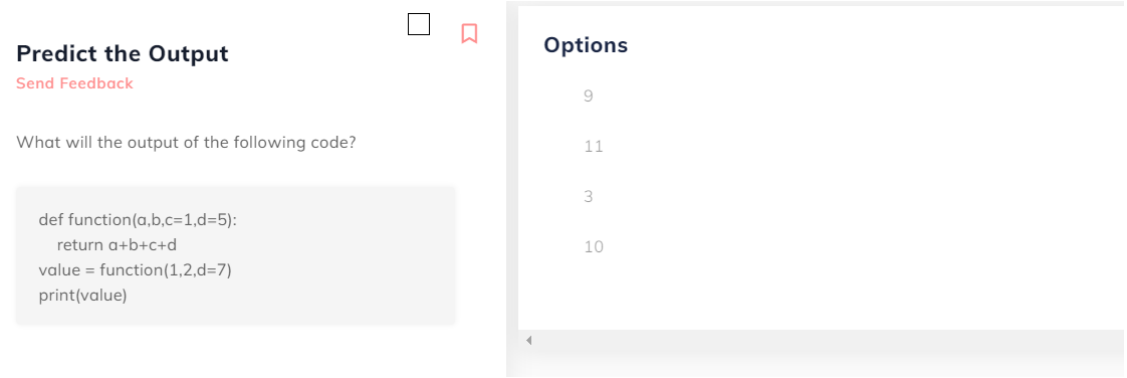
Quiz



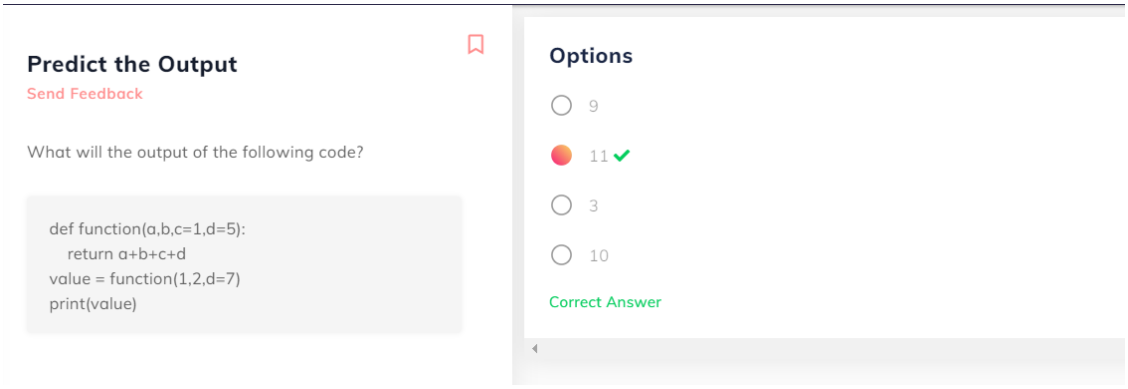
Solution



Quiz



Solution



Mini Project Snake-Water-Gun Game Development

Problem statement is you have to develope a snake water gun game. Rules are mentioned below.
Snake>Water
Water>Gun
Gun>Snake

In [97]:

```
import random

game_list = ["Snake","Water","Gun"]
user_score=0
computer_score=0
name=input("Enter your name: ")
print("{} Let's start the game".format(name))
turn=1
while(turn<=5):
    print()
    print("Enter your choice: ")
    print("1. Snake")
    print("2. Water")
    print("3. Gun")
    print()

    print("*********************************************{} turn******
    computer_choice=random.choice(game_list)
    user_choice=int(input("Enter your choice: "))
    user_choice=game_list[user_choice-1]

    if(user_choice=="Snake" and computer_choice=="Snake"):
        print("Game draw")
        print("{} your choice {}".format(name,user_choice))
        print("computer choice {}".format(computer_choice))
        print("{} score is {}".format(name,user_score))
        print("Computer score is {}".format(computer_score))
        turn=turn+1
    elif(user_choice=="Water" and computer_choice=="Water"):
        print("Game draw")
        print("{} your choice {}".format(name,user_choice))
        print("computer choice {}".format(computer_choice))
        print("{} score is {}".format(name,user_score))
        print("Computer score is {}".format(computer_score))
        turn=turn+1
    elif(user_choice=="Gun" and computer_choice=="Gun"):
        print("Game draw")
        print("{} your choice {}".format(name,user_choice))
```

```
        print("computer choice {}".format(computer_choice))
        print("{} score is {}".format(name,user_score))
        print("Computer score is {}".format(computer_score))
        turn=turn+1


    elif(user_choice=="Snake" and computer_choice=="Water"):
        print("{} your choice {}".format(name,user_choice))
        print("computer choice {}".format(computer_choice))
        user_score=user_score+1
        print("Congratulations! {} you won the game in {} turn".format(na
        print("{} score is {}".format(name,user_score))
        print("Computer score is {}".format(computer_score))
        turn=turn+1
    elif(user_choice=="Water" and computer_choice=="Gun"):
        print("{} your choice {}".format(name,user_choice))
        print("computer choice {}".format(computer_choice))
        user_score=user_score+1
        print("Congratulations! {} you won the game in {} turn".format(na
        print("{} score is {}".format(name,user_score))
        print("Computer score is {}".format(computer_score))
        turn=turn+1
    elif(user_choice=="Gun" and computer_choice=="Snake"):
        print("{} your choice {}".format(name,user_choice))
        print("computer choice {}".format(computer_choice))
        user_score=user_score+1
        print("Congratulations! {} you won the game in {} turn".format(na
        print("{} score is {}".format(name,user_score))
        print("Computer score is {}".format(computer_score))
        turn=turn+1

    elif(user_choice=="Snake" and computer_choice=="Gun"):
        print("{} your choice {}".format(name,user_choice))
        print("computer choice {}".format(computer_choice))
        computer_score=computer_score+1
        print("Oh no! computer won the game in {} turn".format(turn))
        print("{} score is {}".format(name,user_score))
        print("Computer score is {}".format(computer_score))
        turn=turn+1
    elif(user_choice=="Water" and computer_choice=="Snake"):
        print("{} your choice {}".format(name,user_choice))
        print("computer choice {}".format(computer_choice))
        computer_score=computer_score+1
        print("Oh no! computer won the game in {} turn".format(turn))
        print("{} score is {}".format(name,user_score))
        print("Computer score is {}".format(computer_score))
        turn=turn+1
    elif(user_choice=="Gun" and computer_choice=="Water"):
        print("{} your choice {}".format(name,user_choice))
        print("computer choice {}".format(computer_choice))
        computer_score=computer_score+1
        print("Oh no! computer won the game in {} turn".format(turn))
        print("{} score is {}".format(name,user_score))
        print("Computer score is {}".format(computer_score))
        turn=turn+1
    else:
        print("Invalid character blocked")

print()
print()
print("**********************************************Final Result*****
if(user_score>computer_score):
    print("{} you have won the game".format(name))
    print("Your final score is: ",user_score)
```

```
    print("Computer final score is: ",computer_score)
elif(user_score<computer_score):
    print("{} you have loss the game".format(name))
    print("Your final score is: ",user_score)
    print("Computer final score is: ",computer_score)
elif(user_score==computer_score):
    print("Game is draw")
    print("Your final score is: ",user_score)
    print("Computer final score is: ",computer_score)
```

```
Enter your name: sachin kapoor
sachin kapoor Let's start the game

Enter your choice:
1. Snake
2. Water
3. Gun

**********************************************1 turn*******************
**************************
Enter your choice: 1
Game draw
sachin kapoor your choice Snake
computer choice Snake
sachin kapoor score is 0
Computer score is 0

Enter your choice:
1. Snake
2. Water
3. Gun

**********************************************2 turn*******************
**************************
Enter your choice: 2
sachin kapoor your choice Water
computer choice Snake
Oh no! computer won the game in 2 turn
sachin kapoor score is 0
Computer score is 1

Enter your choice:
1. Snake
2. Water
3. Gun

**********************************************3 turn*******************
**************************
Enter your choice: 3
sachin kapoor your choice Gun
computer choice Water
Oh no! computer won the game in 3 turn
sachin kapoor score is 0
Computer score is 2

Enter your choice:
1. Snake
2. Water
3. Gun

**********************************************4 turn*******************
**************************
Enter your choice: 1
sachin kapoor your choice Snake
computer choice Gun
Oh no! computer won the game in 4 turn
sachin kapoor score is 0
Computer score is 3
```

```
Enter your choice:
1. Snake
2. Water
3. Gun

**********************************************5 turn******************
**************************
Enter your choice: 2
sachin kapoor your choice Water
computer choice Snake
Oh no! computer won the game in 5 turn
sachin kapoor score is 0
Computer score is 4


**********************************************Final Result************
*********************************
sachin kapoor you have loss the game
Your final score is:  0
Computer final score is:  4
```

==========

Strings:

Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes. The computer does not understand the characters; internally, it stores manipulated character as the combination of the 0's and 1's.

Each character is encoded in the ASCII or Unicode character. So we can say that Python strings are also called the collection of Unicode characters.

In Python, strings can be created by enclosing the character or the sequence of characters in the quotes. Python allows us to use single quotes, double quotes, or triple quotes to create the string.

How to create a string: Strings can be created by enclosing a sequence of characters inside a single quote or double-quotes.

Example

In [98]:
```
message="Good Morning"
print(message)

message='Good Morning'
print(message)
```

```
Good Morning
Good Morning
```

Strings are imutable: Immutable means you can not make changes in string after initializing it. We can verify this by trying to update a part of the string which will led us to an error.

In [99]:
```
t= "Tutorialspoint"
print(type(t))
t[0] = "M"
```

```
<class 'str'>
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-99-e51930b5ace7> in <module>
      1 t= "Tutorialspoint"
      2 print(type(t))
----> 3 t[0] = "M"

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

Code Explain: When we run the above program, we get the following output −

t[0] = "M"
TypeError: 'str' object does not support item assignment

We can further verify this by checking the memory location address of the position of the letters of the string.

In [100…
```
x = 'banana'

for idx in range (0,6):
    print(x[idx], "=", id(x[idx]))
```

```
b = 2310306661168
a = 2310306685168
n = 2310306548144
a = 2310306685168
n = 2310306548144
a = 2310306685168
```

When we run the above program we get the following output. As you can see above 2nd character "a", 4th character "a" and 6th character "a" point to same location. Also "n" and "n" also point to the same location.

Accessing characters in a string: String indices start at 0 and go on till 1 less than the length of the string. We can use the index operator [ ] to access a particular character in a string. Eg.

Index:	0	1	2	3	4
String "hello":	"h"	"e"	"l"	"l"	"o"

NOTE

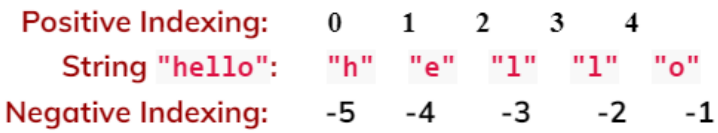
Trying to access indexes out of the range (0,lengthOfString-1), will raise an IndexError. Also, the index must be an integer. We can't use float or other data types;this will result in TypeError.

Let us take an example to understand how to access characters in a String:

In [101…
```
s= "hello"
print(s[0]) #Output: h
print(s[2]) #Output: l
print(s[4]) #Output: o
```

```
h
l
o
```

Negative indexing: Python allows negative indexing for strings. The index of -1 refers to the last character, -2 to the second last character, and so on. The negative indexing starts from the last character in the string.



Let us take an example to understand how to access characters using negative indexing in a string:

In [102…
```
s= "hello"
print(s[-1]) #Output: o
print(s[-2]) #Output: l
print(s[-3]) #Output: l
```

```
o
l
l
```

Concatenation: Joining of two or more strings into a single string is called string concatenation. The + operator does this in Python.

Example

In [103…
```
# Python String Operations
str1 = 'Hello'
str2 ='World!'
# using +
print('str1 + str2 = ', str1 + str2)
```

```
str1 + str2 =  HelloWorld!
```

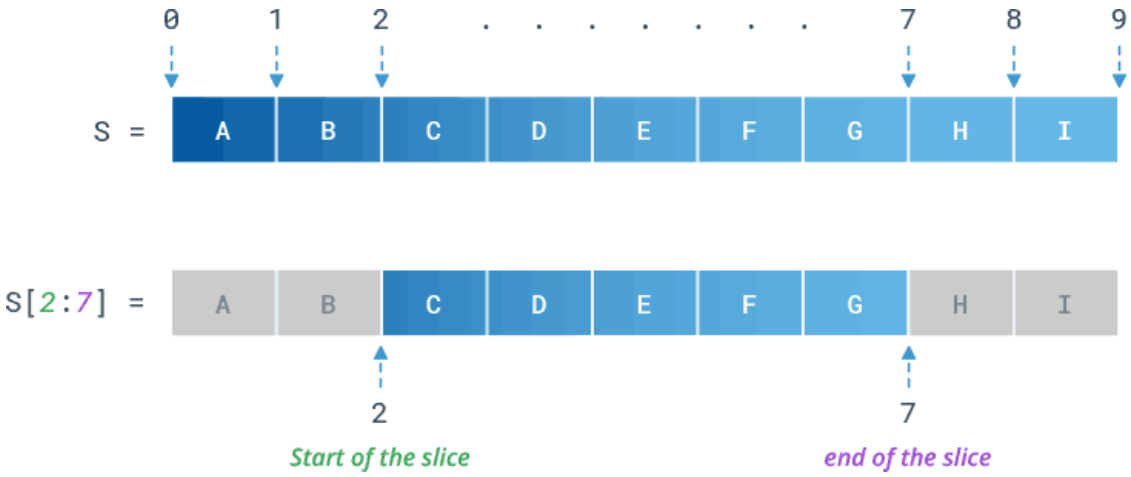
Slicing: String slicing refers to accessing a specific portion or a subset of a string with the original string remaining unaffected. You can use the indexes of string characters to create string slices as per the following syntax:

slice= [StartIndex : StopIndex : Steps]

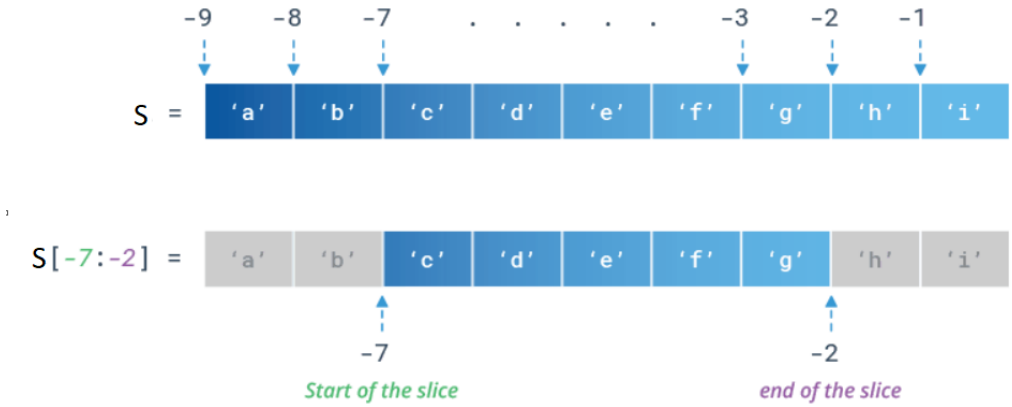
- The StartIndex represents the index from where the string slicing is supposed to begin. Its default value is 0, i.e. the string begins from index 0 if no StartIndex is specified.
- The StopIndex represents the last index up to which the string slicing will go on. Its default value is (length(string)-1) or the index of the last character in the string.
- steps represent the number of steps. It is an optional parameter. steps, if defined, specifies the number of characters to jump over while counting from StartIndex to StopIndex. By default, it is 1.
- The string slices created, include characters falling between the indexes StartIndex and StopIndex, including StartIndex and not including StopIndex.

In [104…
```
S="abcdefghi"
print(S[2:7])
```

```
cdefg
```



Slice using negative slicing: You can also specify negative indices while slicing a string. Consider the example given below.



In [105…
```
S="abcdefghi"
print(S[-7:-2])
```

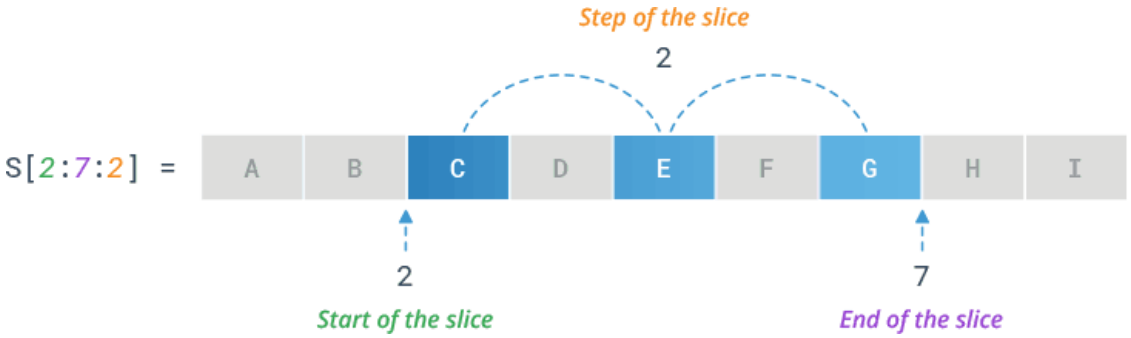
```
cdefg
```

Specify Step of the Slicing

You can specify the step of the slicing using the steps parameter. The steps parameter is optional and by default 1.

In [106…
```
S="abcdefghi"
print(S[2:7:2])
```

```
ceg
```



Length: To get the length of a string, use the len() function

Example

In [107…
```
message="Good Morning"
```

```
print("The length of message is: ",len(message))
```

```
The length of message is:  12
```

Strip: Method removes whitespaces from the begining to end

Example

In [108…
```
message="  Good Morning   "
print("After stripping of message: ",message.strip())
```

```
After stripping of message:  Good Morning
```

rStrip: Method removes whitespaces only from the right

Example

In [109…
```
message="  Good Morning   "
print("After right stripping of message: ",message.rstrip())
```

```
After right stripping of message:    Good Morning
```

lStrip: Method removes whitespaces only from the left

Example

In [110…
```
message="  Good Morning   "
print("After right stripping of message: ",message.lstrip())
```

```
After right stripping of message:  Good Morning
```

Lower: The lower() method returns the string in lower case.

Example

In [111…
```
message="Good Morning"
print("After converting into lowercase: ",message.lower())
```

```
After converting into lowercase:  good morning
```

Upper: The upper() method returns the string in upper case.

Example

In [112…
```
message="Good Morning"
print("After converting into uppercase: ",message.upper())
```

```
After converting into uppercase:  GOOD MORNING
```

replace(): The replace() method replaces a string with another string.

Example

In [113…
```
message="Hello, World!"
print(message.replace("H","F"))
```

```
Fello, World!
```

split(): The split() method splits the string into substring if it finds instances of the seperator.

Example

In [114…
```
message="Hello, World!"
print(message.split(" "))
```

```
['Hello,', 'World!']
```

capitalize(): Converts the first character to upper case.

Example

In [115…
```
message="hello, World!"
print(message.capitalize())
```

```
Hello, world!
```

center(): The center() method will center align the string, using a specified character(space is default) as the fill character.

Example

In [116…
```
fruit="Banana"
print(fruit.center(20,"O"))
```

```
OOOOOOOBananaOOOOOOO
```

count(): Return the number of times a specified value occurs in a string.

Example

In [117…
```
fruit="Banana"
print(fruit.count("a"))
```

```
3
```

endswith(): Returns True if the string ends with the specified value, otherwise False

Example

In [118…
```
message="Hello, World!"
print(message.endswith("World!"))

message="Hello, World!"
print(message.endswith("Hello"))
```

```
True
False
```

find(): Searhes the string for a specified values and returns the position of where it was found

Example

In [119…
```
message="Hello, World!"
print(message.find("World!"))
```

```
7
```

NOTE: The index() method is almost the same as the find() method, the only difference is that the find() method returns -1 if the value is not foundor index() method raise an exception if the value is not found

isalnum(): Return True if all characters in the strings are alphanumeric

Example

In [120…
```
message="HelloWorld"
print(message.isalnum())

message="Hello World"
print(message.isalnum())
```

```
True
False
```

isalpha(): Return True if all characters in the strings are in the alphabet

Example

In [121…
```
message="HelloWorld"
print(message.isalpha())

message="HelloWorld1"
print(message.isalpha())
```

```
True
False
```

isdecimal(): Return True if all characters in the strings are in the decimal

Example

In [122…
```
message="14.23"
print(message.isdecimal())
```

```
False
```

isdigit(): Return True if all characters in the strings are in the digit

Example

In [123…
```
message="141"
print(message.isdigit())

message="Python"
print(message.isdigit())
```

```
True
False
```

isnumeric(): Return True if all characters in the strings are in the numeric

Example

In [124…
```
message="141"
print(message.isnumeric())

message="Python"
print(message.isnumeric())
```

```
True
False
```

islower(): Return True if all characters in the strings are in the lower case

Example

In [125…
```
message="sachin"
print(message.islower())

message="SACHIN"
print(message.islower())
```

```
True
False
```

isupper(): Return True if all characters in the strings are in the uppercase

Example

In [126…
```
message="SACHIN"
print(message.isupper())
```

```
message="sachin"
print(message.isupper())
```

```
True
False
```

istitle(): This istitle() method returns True if all words in a text start with a upper case letter, and the rest of the word are lowercase letter, otherwise False

Example

In [127…
```
message="Sachin Kapoor"
print(message.istitle())
```

```
True
```

Partition(): The partition method searches for a string, and splits the string into a tuple containing three elements. The first element contains the part before the specified string. The second element contains the specified string. The third element contains the part after the string

Example

In [128…
```
message="I could eat bananas all day"
print(message.partition("bananas"))
```

```
('I could eat ', 'bananas', ' all day')
```

Comparing Strings:

1. In string comparison, we aim to identify whether two strings are equivalent to each other and if not, which one is greater.
2. String comparison in Python takes place character by character. That is, characters in the same positions are compared from both the strings.
3. If the characters fulfill the given comparison condition, it moves to the characters in the next position. Otherwise, it merely returns False .
    NOTE: Some points to remember when using string comparison operators:
   - The comparisons are case-sensitive , hence same letters in different letter cases(upper/lower) will be treated as separate characters.
   - If two characters are different, then their Unicode value is compared; the character with the smaller Unicode value is considered to be lower.

   This is done using the following operators:
   - == : This checks whether two strings are equal.
   - != : This checks if two strings are not equal.
   - < : This checks if the string on its left is smaller than that on its right.
   - <= : This checks if the string on its left is smaller than or equal to that on its right.
   - > : This checks if the string on its left is greater than that on its right.
   - >= : This checks if the string on its left is greater than or equal to that on its right.

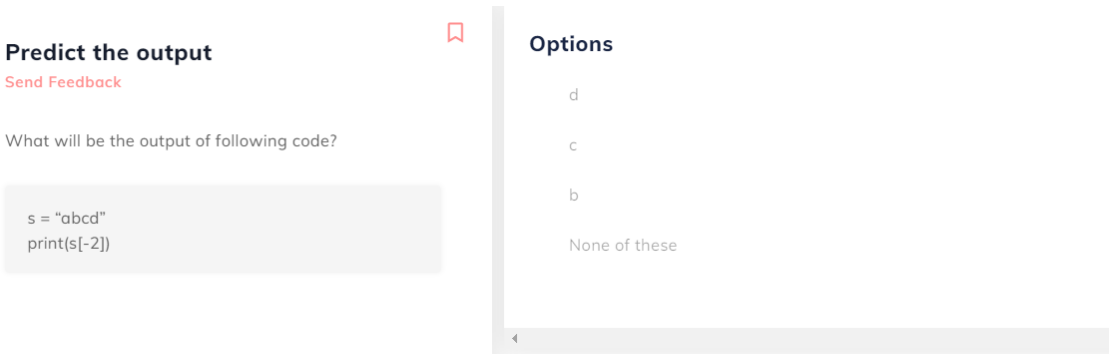
Example:

In [129…
```
str1="Sachin kapoor"
str2="Ranjeet kumar singh"
str3="Sachin kapoor"
```

```
if(str1==str2):
    print("String1 and String 2 are same")
else:
    print("String1 and String 2 are different")

if(str1==str3):
    print("String1 and String 3 are same")
else:
    print("String1 and String 3 are different")

if(str1!=str2):
    print("String1 and String 2 are different")
else:
    print("String1 and String 2 are samet")

if(str1!=str3):
    print("String1 and String 3 are different")
else:
    print("String1 and String 3 are same")

if(str1>str2):
    print("String1 is greater then String2")
else:
    print("String1 is smaller then String2")

if(str1>str3):
    print("String1 is greater then String3")
else:
    print("String1 is smaller then String3")

if(str2>str3):
    print("String2 is greater then String3")
else:
    print("String2 is smaller then String3")
```

```
String1 and String 2 are different
String1 and String 3 are same
String1 and String 2 are different
String1 and String 3 are same
String1 is greater then String2
String1 is smaller then String3
String2 is smaller then String3
```

Quiz



Solution

Predict the output

Send Feedback

What will be the output of following code?

```
s = “abcd”
print(s[-2])
```

Options

- d
- c ✔
- b
- None of these

Correct Answer

Quiz

Predict the output

Send Feedback

What will the output of following code?

```
s="abcd"
s[0]='c'
print(s)
```

Options

- cbcd
- abcd
- Error
- None of these

Solution

Predict the output

Send Feedback

What will the output of following code?

```
s="abcd"
s[0]='c'
print(s)
```

Options

- cbcd
- abcd
- Error ✔
- None of these

Correct Answer

Quiz

Predict the output

Send Feedback

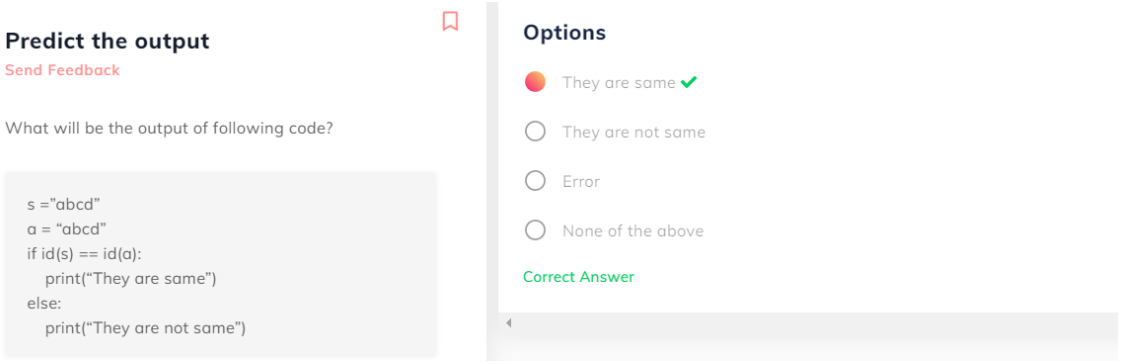
What will be the output of following code?

```
s =”abcd”
a = “abcd”
if id(s) == id(a):
   print(“They are same”)
else:
   print(“They are not same”)
```

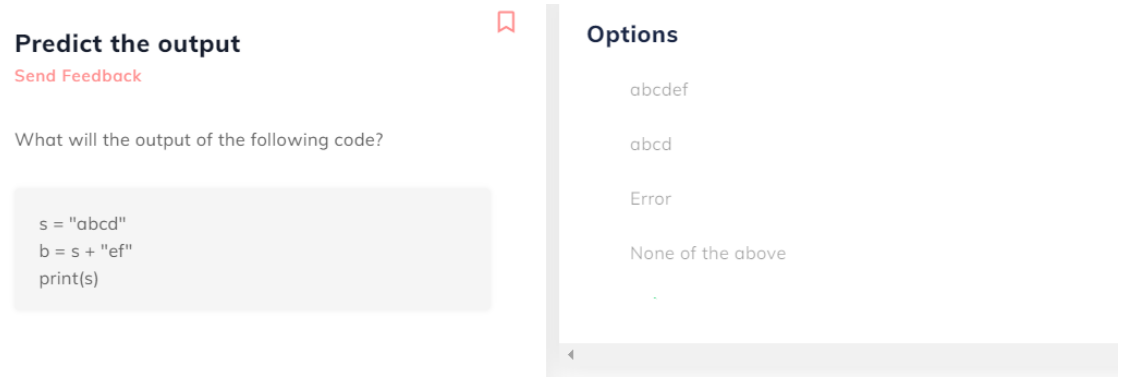
Options

- They are same
- They are not same
- Error
- None of the above

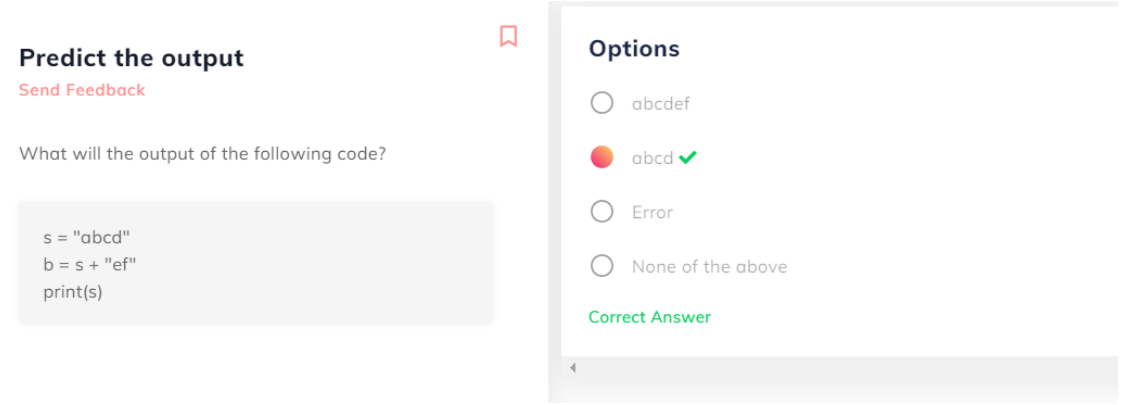
Solution



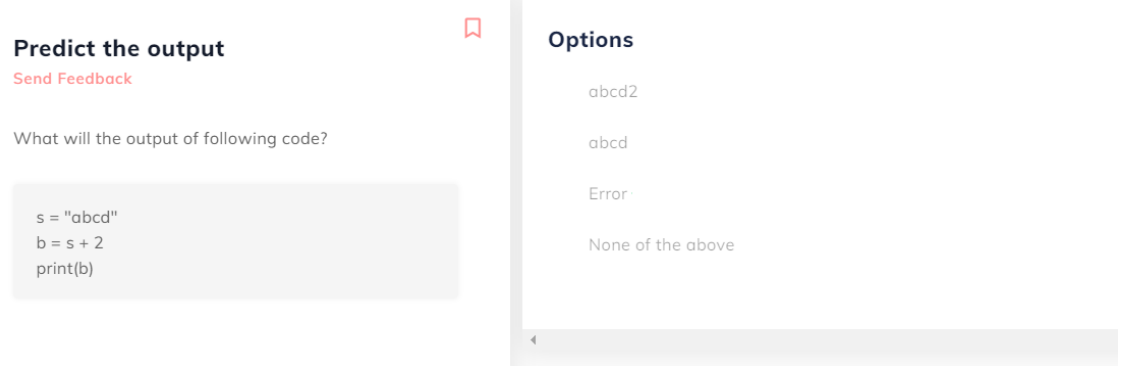
Quiz



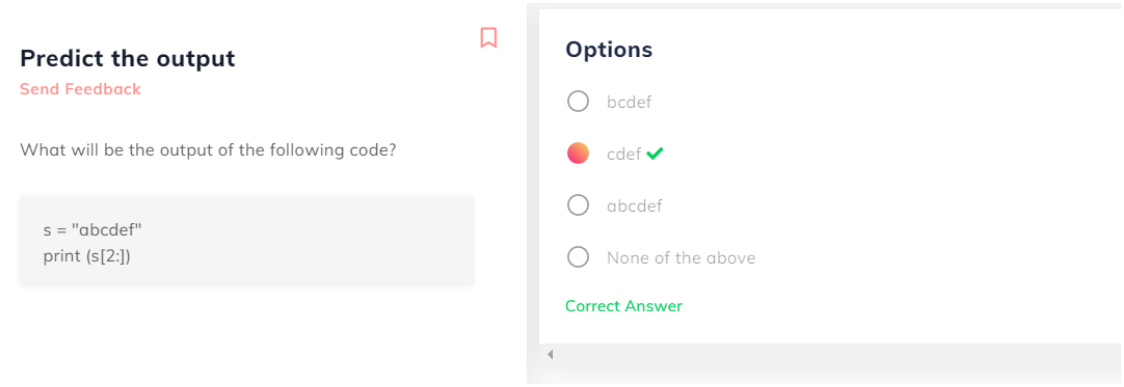
Solution



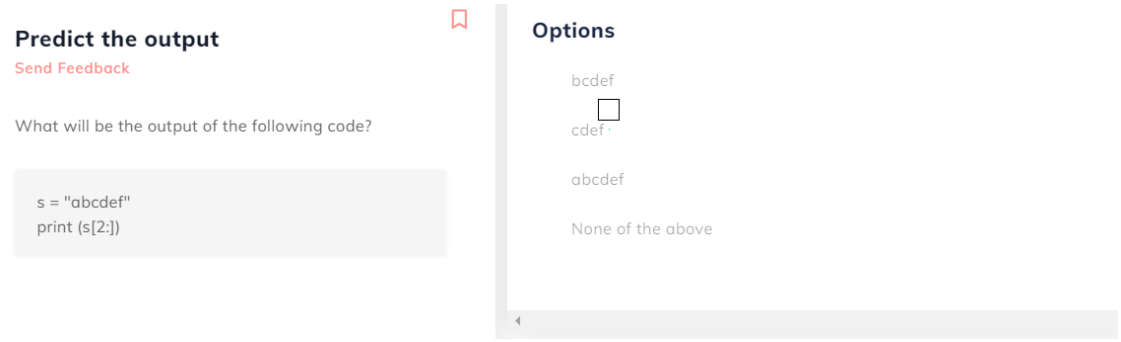
Quiz



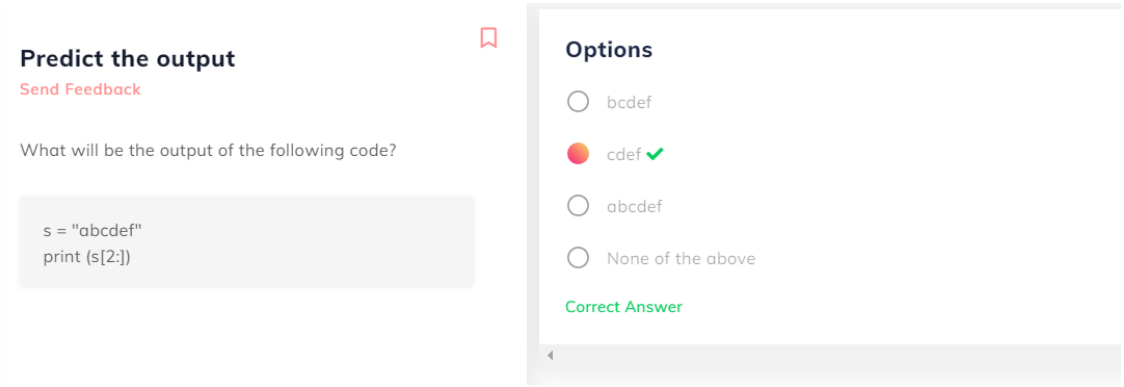
Solution



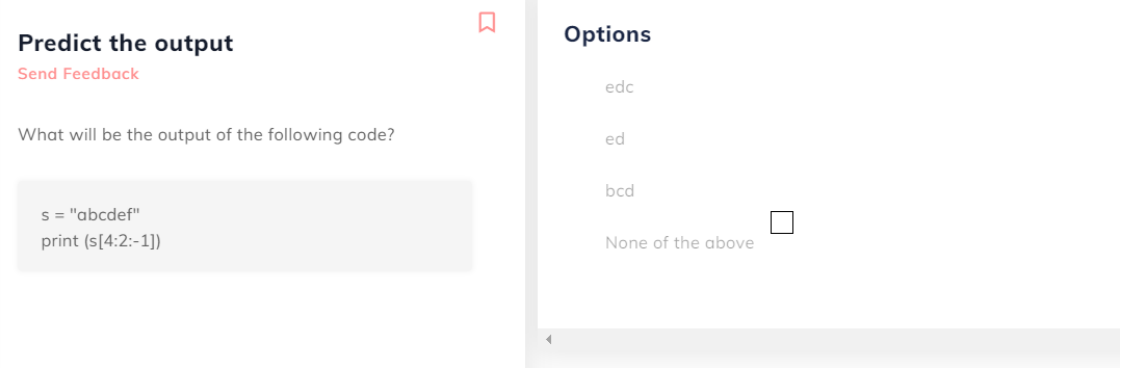
Quiz



Solution



Quiz



Solution

Predict the output

Send Feedback

What will be the output of the following code?

```
s = "abcdef"
print (s[4:2:-1])
```

Options

- edc
- ed ✔
- bcd
- None of the above

Correct Answer

Quiz

Predict the output

Send Feedback

What will be the output of the following code?

```
a = "abcdef" == "abcd"
print(a)
```

Options

- False
- True
- Error
- None of the above

Solution

Predict the output

Send Feedback

What will be the output of the following code?

```
a = "abcdef" == "abcd"
print(a)
```

Options

- False ✔
- True
- Error
- None of the above

Correct Answer

Quiz

Predict the output

Send Feedback

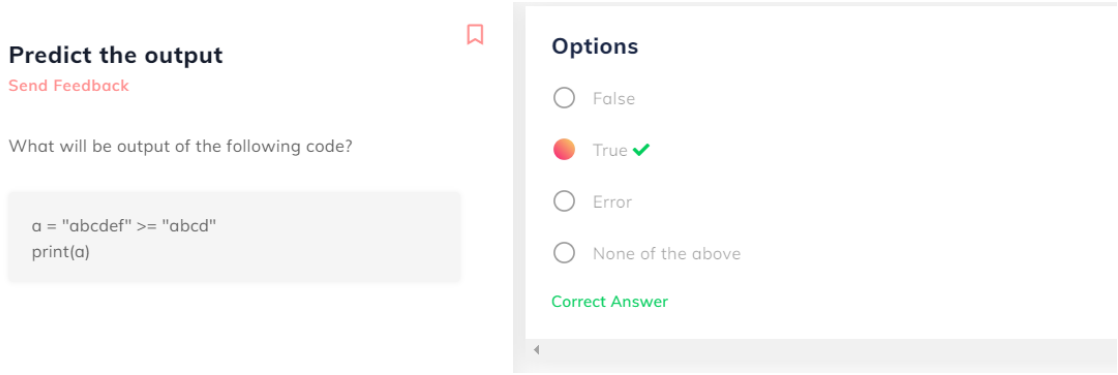
What will be output of the following code?

```
a = "abcdef" >= "abcd"
print(a)
```

Options

- False
- True
- Error
- None of the above

Solution



==========

List

list is a collection of arbitrary objects, somewhat akin to an array in many other programming languages but more flexible. Lists are defined in Python by enclosing a comma-separated sequence of objects in square brackets ([])

The important characteristics of Python lists are as follows:

- Lists are ordered.
- Lists can contain any arbitrary objects.
- List elements can be accessed by index.
- Lists are mutable.

Creating Lists: In Python programming, a list is created by placing all the items (elements) inside square brackets [ ], separated by commas.
It can have any number of items and they may be of different types (integer, float, string etc.).

list1 = [] #Empty list
list2 = [1, 2, 3] #List of integers
list3 = [1, "One", 3.4] #List with mixed data types

A list can also have another list as an element. Such a list is called a Nested List.
list4 = ["One", [8, 4, 6], ['Three']] #Nested List

Lists are ordered: A list is not only a collection of objects. It is an ordered collection of objects. The order in which you specify the elements when you define a list is an innate characteristic of that list and is maintained for that list's lifetime.

Example

In [130…
```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
```

Lists Can Contain Arbitrary Objects: A list can contain any assortment of objects. The elements of a list can all be the same type or the element can be of varying type

Example

In [131…
```
li=[2,4,6,8]
print(li)
```

```
[2, 4, 6, 8]
```

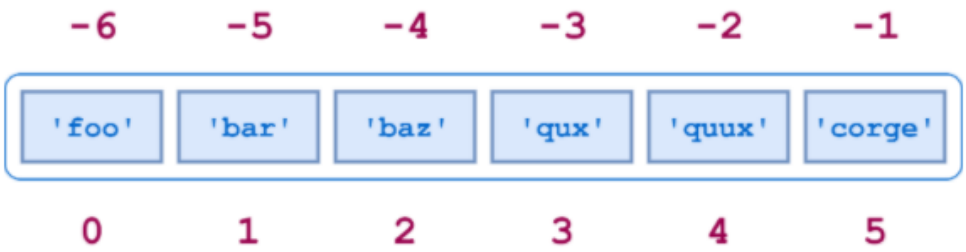
In [132…
```
li=[1,2,"python",3.5]
print(li)
```

```
[1, 2, 'python', 3.5]
```

List Elements Can Be Accessed by Index: Individual elements in a list can be accessed using an index in square brackets. Indexing start from 0 and negtive indexing start from back.

Example

In [133…
```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
```



Negative List Indexing

In [134…
```
print(a[0])
print(a[1])
print(a[2])
print(a[3])
print(a[-1])
print(a[-2])
print(a[-3])
print(a[-4])
```

```
foo
bar
baz
qux
corge
quux
qux
baz
```

Lists Are Mutable: What do you mean by mutable. Mutable means you can change the values of list. Let us deep dive into the concept of mutable and immutbale what is this. Let us understand with example:

As we all know that when we create list. The list only stores the address where the actual content stored and that list name start pointing towards that address. In list if we try to change anything on list it will make changes on original list.

In [135…
```
li=[1,2,3,4,5]
```

In [136…
```
li2=li
```

In [137…
```
li2[3]=0
print(li)
print(li2)
```

```
[1, 2, 3, 0, 5]
[1, 2, 3, 0, 5]
```

As you can see when we declare list **"li"** start pointing to the list and when we initialize **"li2"** with **"li"** then **"li2"** also start pointing to the same list. If we try to edit list via **"li2"** it will reflect to **"li"** as well because both are pointing to the same list.

Example

In Python, square brackets ([]) indicate a list, and individual elements in the list are separated by commas. Here's a simple example of a list that contains a few kinds of bicycles:

In [138…
```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
```

Accessing Elements in a List: you can access any element in a list by telling Python the position, or index, of the item desired.

Example

In [139…
```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0])
```

```
trek
```

Modifying Elements in a List:

Example

In [140…
```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles[0] = 'ducati'
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
['ducati', 'yamaha', 'suzuki']
```

List slicing: List slicing refers to accessing a specific portion or a subset of a list while the original list remains unaffected. You can use indexes of list elements to create list slices as per the following syntax:

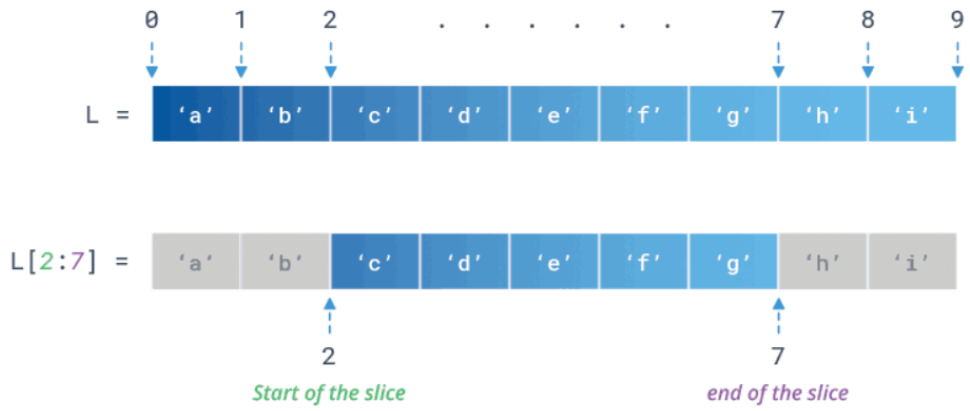
slice= [StartIndex : StopIndex : Steps]

- The StartIndex represents the index from where the list slicing is supposed to begin. Its default value is 0, i.e. the list begins from index 0 if no StartIndex is specified.
- The StopIndex represents the last index up to which the list slicing will go on. Its default value is (length(list)-1) or the index of the last element in the list.
- steps represent the number of steps. It is an optional parameter. steps, if defined, specifies the number of elements to jump over while counting from StartIndex to StopIndex. By default, it is 1.
- The list slices created, include elements falling between the indexes StartIndex and StopIndex, including StartIndex and not including StopIndex.

Example

In [141…
```
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[2:7])
```

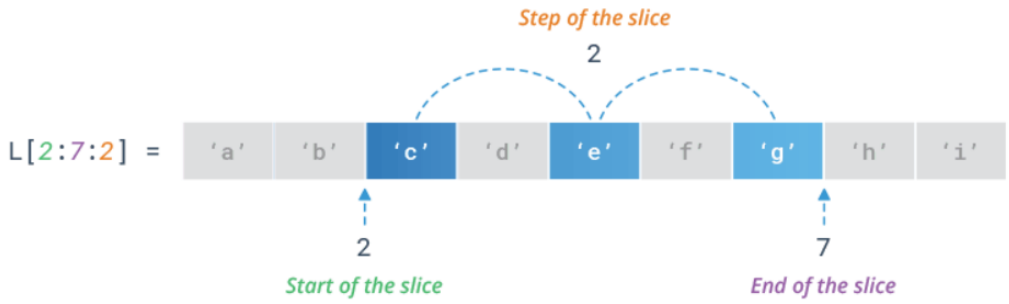
```
['c', 'd', 'e', 'f', 'g']
```



Specify Step of the Slicing: You can specify the step of the slicing using the steps parameter. The steps parameter is optional and by default 1.

In [142…
```
# Print every 2nd item between position 2 to 7
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[2:7:2])
```

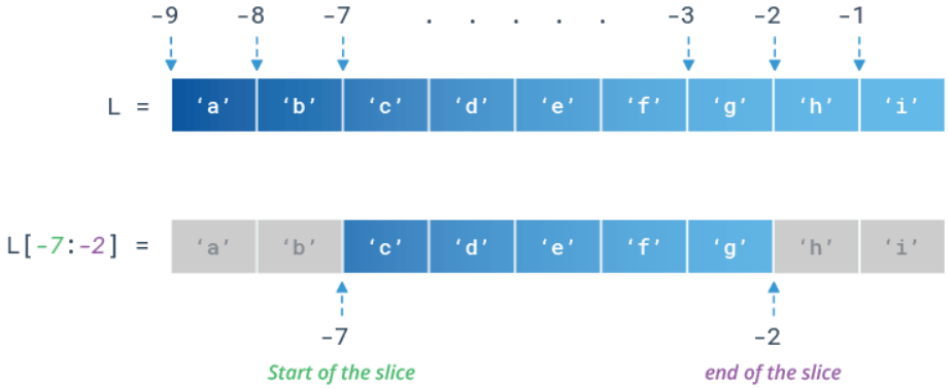
```
['c', 'e', 'g']
```



You can even specify a negative step size:

In [143…
```
# Print every 2nd item between position 6 to 1
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[6:1:-2])
```

```
['g', 'e', 'c']
```



len(): To determine how many items in a list has use the len() function

Example

In [144…
```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print("The length of the list is : ",len(bicycles))
```

```
The length of the list is :  4
```

append(): To add an item to the end of the list, use the append() method

Example

In [145…
```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)

motorcycles.append('ducati')
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha', 'suzuki', 'ducati']
```

insert(): To add an item at the specified index, use the insert() method

Example

In [146…
```
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.insert(0, 'ducati')
print(motorcycles)
```

```
['ducati', 'honda', 'yamaha', 'suzuki']
```

del: The del keyword removes the specified index

Example

In [147…
```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)

del motorcycles[0]
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
['yamaha', 'suzuki']
```

pop(): The pop() method removes the last item in a list, but it lets you work with that item after removing it.

Example

In [148…
```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
popped_motorcycle = motorcycles.pop()

print(motorcycles)
print(popped_motorcycle)
```

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha']
suzuki
```

Popping Items from any Position in a List: You can actually use pop() to remove an item in a list at any position by including the index of the item you want to remove in parentheses.

Example

In [149…
```
first_owned = motorcycles.pop(0)
print('The first motorcycle I owned was a ' + first_owned.title() + '.')
```

```
The first motorcycle I owned was a Honda.
```

remove(): Sometimes you won't know the position of the value you want to remove from a list. If you only know the value of the item you want to remove, you can use the remove() method.

Example

In [150…
```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
print(motorcycles)

motorcycles.remove('ducati')
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
```

NOTE: The remove() method deletes only the first occurrence of the value you specify. If there'is a possibility the value appears more than once in the list, you'll need to use a loop to determine if all occurrences of the value have been removed.

sort(): sort() method is used to sort the list.
The sort() method, shown at u, changes the order of the list permanently.

Example

In [151…
```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort()
print(cars)
```

```
['audi', 'bmw', 'subaru', 'toyota']
```

You can also sort this list in reverse alphabetical order by passing the argument reverse=True to the sort() method. The following example sorts the list of cars in reverse alphabetical order:

In [152…
```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort(reverse=True)
print(cars)
```

```
['toyota', 'subaru', 'bmw', 'audi']
```

sorted(): To maintain the original order of a list but present it in a sorted order, you can use the sorted() function. The sorted() function lets you display your list in a particular order but doesn't affect the actual order of the list.

Example

In [153…
```
cars = ['bmw', 'audi', 'toyota', 'subaru']

print("Here is the original list:")
print(cars)

print("\nHere is the sorted list:")
print(sorted(cars))

print("\nHere is the original list again:")
print(cars)
```

```
Here is the original list:
['bmw', 'audi', 'toyota', 'subaru']

Here is the sorted list:
['audi', 'bmw', 'subaru', 'toyota']

Here is the original list again:
['bmw', 'audi', 'toyota', 'subaru']
```

reverse(): To reverse the original order of a list, you can use the reverse() method.

Example

In [154…
```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars.reverse()
print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['subaru', 'toyota', 'audi', 'bmw']
```

clear(): The clear() method empties the list

Example

In [155…
```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.clear()
print(cars)
```

```
[]
```

copy(): You can not copy a list by typing list2=list1 because list2 will only be a reference

to list1, and changes made in list1 will automatically also be made in list2

Example

In [156…
```
cars = ['bmw', 'audi', 'toyota', 'subaru']
latestcars=cars.copy()
print(cars)
print(latestcars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['bmw', 'audi', 'toyota', 'subaru']
```

list() constructor: It is also possible to use the list() constructor to make a new list.

Example

In [157…
```
cars=[]
print(cars)
print(type(cars))
```

```
[]
<class 'list'>
```

counts(): Returns the numbers of element with the specified value.

Example

In [158…
```
cars = ['bmw', 'audi', 'toyota', 'subaru','bmw','audi']
print(cars.count('toyota'))
print(cars.count('bmw'))
```

```
1
2
```

index(): Return the index of the first element with the specified value.

Example

In [159…
```
cars = ['bmw', 'audi', 'toyota', 'subaru','bmw','audi']
print(cars.index('audi'))
```

```
1
```

max(): max() function give you the maximum value present in the list.

Example

In [160…
```
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
print(max(digits))
```

```
9
```

min(): min() function give you the minimum value present in the list.

Example

In [161…
```
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
print(min(digits))
```

```
0
```

2 Dimension list

As we learn list. We also have a concept of 2 Dimension list. First of all we need to think. Why we need 2D list or what is the need of 2D list.

Supoose you need to work on matrix. In matrix we have some row and some column. So how we can implement matrix through list. Is it possible?
It is possible but too complex.

- A Two-dimensional list is a list of lists.
- It represents a table/matrix with rows and columns of data.
- In this type of list, the position of a data element is referred to by two indices instead of one.

Declaration of 2D array: 2D list is nothing just list of list.

Example

In [164…
```
arr=[[1,2,3],[4,5,6],[7,8,9]]
print("2D list are: ",arr)
```

```
2D list are:  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Take input of 2D list: Input to a 2-D array is provided in the form of rows and columns.

Example

In [166…
```
row=int(input("Enter the number of row: "))
col=int(input("Enter the number of column: "))
arr=[]

for r in range(row):
    newarr=[]
    for c in range(col):
        item=int(input())
        newarr.append(item)
    arr.append(newarr)

print("2D list are: ",arr)
```

```
Enter the number of row: 5
Enter the number of column: 2
1
2
3
4
5
6
7
8
9
10
2D list are:  [[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]]
```

Accessing Values in a Two Dimensional list: The data elements in two-dimensional lists can be accessed using two indices. One index referring to the main or parent list (inner list) and another index referring to the position of the data element in the inner list. If we mention only one index then the entire inner list is printed for that index position.

Example:

In [167…
```
T = [[ 11 , 12 , 5 , 2 ], [ 15 , 6 , 10, 6 ], [ 10 , 8 , 12 , 5 ], [ 12
print(T[ 0 ]) #List at index 0 in T
print(T[ 1 ][ 2 ]) #Element at index 2 in list at index 1 in T
```

```
[11, 12, 5, 2]
10
```

Updating Values in Two Dimensional list: We can update the entire inner list or some specific data elements of the inner list by reassigning the values using the list index.

Example:

In [168…
```
T = [[ 11 , 12 , 5 , 2 ], [ 15 , 6 , 10, 6 ], [ 10 , 8 , 12 , 5 ], [ 12
print("T :",T)

T[ 0 ][ 1 ]= 1 #Update the element at index 1 of list at index 0 of T to
T[ 1 ][ 1 ]= 7

print("After updating T value: ",T)
```

```
T : [[11, 12, 5, 2], [15, 6, 10, 6], [10, 8, 12, 5], [12, 15, 8, 6]]
After updating T value:  [[11, 1, 5, 2], [15, 7, 10, 6], [10, 8, 12, 5],
[12, 15, 8, 6]]
```

Jagged list: As we saw what is 2D list. In 2D list we have equal number of row and column. But in jagged list column size need not to be same.

Example

In [169…
```
jag_list=[[1,2,3],[4,5],[6,7,8],[9]]

print("The jagged list is: ",jag_list)
```

```
The jagged list is:  [[1, 2, 3], [4, 5], [6, 7, 8], [9]]
```

Iterating on 2D list: First we have to iterate a row then all column in particular row and then next row and column of that row and so on.

Example

In [170…
```
arr=[[1,2,3],[4,5,6],[7,8,9]]

for r in range(3):
    for c in range(3):
        print(arr[r][c],end=" ")
    print()
```

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

Looping through an entire list:

Let's say we have a list of magicians' names, and we want to print out each name in the list. We could do this by retrieving each name from the list individually, but this approach could cause several problems. For one, it would be repetitive to do this with a long list of names. Also, we'd have to change our code each time the list's length changed. A for loop avoids both of these issues by letting Python manage these issues internally. Let's use a for loop to print out each name in a list of magicians:

Example

In [171…
```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(magician)
```

```
alice
david
carolina
```

Explain code: We begin by defining a list at u, just as we did in Chapter 3. At v, we define a for loop. This line tells Python to pull a name from the list magicians, and store it in the variable magician. At w we tell Python to print the name that was just stored in magician. Python then repeats lines v and w, once for each name in the list. It might help to read this code as “For every magician in the list of magicians, print the magician’s name.” The output is a simple printout of each name in the list

List comprehension:

List comprehensions are used for creating new lists from other iterable sequences. As list comprehensions return lists, they consist of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element.

Syntax:

In [ ]:
```
output_list = [output_exp for var in input_list if (var satisfies this c
```

NOTE: list comprehension may or may not contain an if condition. List comprehensions can contain multiple for (nested list comprehensions).

Example

In [172…
```
numbers = [ 1 , 2 , 3 , 4 ]
squares = [n** 2 for n in numbers]
print(squares)
```

```
[1, 4, 9, 16]
```

Here, square brackets signify that the output is a list. n **2 is the expression executed for each element and for n in numbers is used to iterate over each element. In other words, we execute n** 2 (expression) for each element in numbers .

Key Points to Remember

- List comprehension is an elegant way to define and create lists based on existing lists.
- List comprehension is generally more compact and faster than normal functions and loops for creating list.
- However, we should avoid writing very long list comprehensions in one line to ensure that code is user-friendly.
- Remember, every list comprehension can be rewritten in for loop, but every for loop can’t be rewritten in the form of list comprehension.

Quiz

Predict the Output

Send Feedback

What will be the output of following code?

```
li =['abcd','def']
li.insert(4,5)
print(li)
```

Options

Index Error

['abcd', 'def']

['abcd', 'def', 5]

None of the above

Solution

Predict the Output

Send Feedback

What will be the output of following code?

```
li =['abcd','def']
li.insert(4,5)
print(li)
```

Options

- Index Error
- ['abcd', 'def']
- ['abcd', 'def', 5] ✔
- None of the above

Correct Answer

Quiz

Predict the Output

Send Feedback

What will be the output of following code?

```
li = ['abcd',5,'def',5]
li.remove(5)
print(li)
```

Options

Error

['abcd', 5, 'def']

['abcd', 'def']

['abcd', 'def', 5]

Solution

Predict the Output

Send Feedback

What will be the output of following code?

```
li = ['abcd',5,'def',5]
li.remove(5)
print(li)
```

Options

- Error
- ['abcd', 5, 'def']
- ['abcd', 'def']
- ['abcd', 'def', 5] ✔

Correct Answer

Quiz

Predict the Output

Send Feedback

What will be the output of following code?

```
li = [5,2,6,8]
li.pop(2)
print(li)
```

Options

[5, 6, 8]

[5, 2, 8]

[5, 2, 6, 8]

Error

Solution

Predict the Output

Send Feedback

What will be the output of following code?

```
li = [5,2,6,8]
li.pop(2)
print(li)
```

Options

- [5, 6, 8]
- [5, 2, 8] ✔
- [5, 2, 6, 8]
- Error

Correct Answer

Quiz

Predict the Output

Send Feedback

What will be the output of follwing code?

```
li = [1,2,3,4,5]
for i in li[1:4]:
    print(i,end= " ")
```

Options

1 2 3 4 5

2 3 4 5

2 3 4

None of the above

Solution

Predict the Output

Send Feedback

What will be the output of follwing code?

```
li = [1,2,3,4,5]
for i in li[1:4]:
    print(i,end= " ")
```

Options

- 1 2 3 4 5
- 2 3 4 5
- 2 3 4 ✔
- None of the above

Correct Answer

Quiz

Predict the Output

Send Feedback

What will be the output of code if following input is provided?

```
5
1
2
3
4
5
n = int(input())
li = []
for i in range(n):
    li.append(input())
print(li)
```

Options

- [1, 2, 3, 4, 5]
- ['1', '2', '3', '4', '5']
- ['12345']
- None of the above

Solution

Predict the Output

Send Feedback

What will be the output of code if following input is provided?

```
5
1
2
3
4
5
n = int(input())
li = []
for i in range(n):
    li.append(input())
print(li)
```

Options

- [1, 2, 3, 4, 5]
- ['1', '2', '3', '4', '5'] ✔
- ['12345']
- None of the above

Correct Answer

Quiz

Predict the Output

Send Feedback

What will be the output of code if following input is provided?

```
1 3 6 8 9
li = [x for x in input().split()]
print(li)
```

Options

- ['1', '3', '6', '8', '9']
- [1, 3, 6, 8, 9]
- ['1 3 6 8 9']
- None of the above

Solution

Predict the Output

Send Feedback

What will be the output of code if following input is provided?

```
1 3 6 8 9
li = [x for x in input().split()]
print(li)
```

Options

- ['1', '3', '6', '8', '9'] ✔
- [1, 3, 6, 8, 9]
- ['1 3 6 8 9']
- None of the above

Correct Answer

Quiz

Predict the Output

Send Feedback

What will be the output of following code?

```
def change(li):
    li[1] = li[1] + 2
li = [1,2,3,4,5]
change(li)
print(li)
```

Options

[3, 2, 3, 4, 5]

[1, 4, 3, 4, 5]

[1, 2, 3, 4, 5]

None of the above

Solution

Predict the Output

Send Feedback

What will be the output of following code?

```
def change(li):
    li[1] = li[1] + 2
li = [1,2,3,4,5]
change(li)
print(li)
```

Options

- [3, 2, 3, 4, 5]
- [1, 4, 3, 4, 5] ✔
- [1, 2, 3, 4, 5]
- None of the above

Correct Answer

Quiz

Predict the Output

Send Feedback

What will be the output of following code?

```
def change(li):
    li[1] = li[1] + 2
    li = [3,3,3,4,5]
li = [1,2,3,4,5]
change(li)
print(li)
```

Options

[3, 2, 3, 4, 5]

[1, 4, 3, 4, 5]

[3, 3, 3, 4, 5]

None of the above ☐

Solution

Predict the Output

Send Feedback

What will be the output of following code?

```
def change(li):
    li[1] = li[1] + 2
    li = [3,3,3,4,5]
li = [1,2,3,4,5]
change(li)
print(li)
```

Options

- [3, 2, 3, 4, 5]
- [1, 4, 3, 4, 5] ✔
- [3, 3, 3, 4, 5]
- None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
li = [[1,2,3],[4,5,6],[7,8,9]]
print(li[2][1])
```

Options

5

8

2

Error

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
li = [[1,2,3],[4,5,6],[7,8,9]]
print(li[2][1])
```

Options

5

8 ✔

2

Error

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
li = [[1,2,3],[4,5,6],[7,8,9]]
print(li[1][3])
```

Options

6

9

3

Error

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
li = [[1,2,3],[4,5,6],[7,8,9]]
print(li[1][3])
```

Options

6

9

3

Error ✔

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
li = [[1,2,3,4],[5,6],[7,8,9]]
print(li[2])
```

Options

[5,6]

Error

[7,8,9]

None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
li = [[1,2,3,4],[5,6],[7,8,9]]
print(li[2])
```

Options

- [5,6]
- Error
- [7,8,9] ✔
- None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
li = [ele**2 for ele in range(5)]
print(li)
```

Options

[1,2,3,4,5]

[1,4,9,16,25]

[0,1,4,9,16]

[0,1,4,9,16,25]

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
li = [ele**2 for ele in range(5)]
print(li)
```

Options

- [1,2,3,4,5]
- [1,4,9,16,25]
- [0,1,4,9,16] ✔
- [0,1,4,9,16,25]

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
li = [ele**2 for ele in range(10) if ele%3 ==0]
print(li)
```

Options

[1,3,9]

[1,9,36,81]

[0,3,6,9]

[0,9,36,81]

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
li = [ele**2 for ele in range(10) if ele%3 ==0]
print(li)
```

Options

- [1,3,9]
- [1,9,36,81]
- [0,3,6,9]
- [0,9,36,81] ✔

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
li = [[ i*j for j in range(4)] for i in range(3)]
print(li)
```

Options

[[0, 0, 0, 0], [0, 1, 2, 3], [0, 2, 4, 6]]

[[0, 0, 0], [0, 1, 2], [0, 2, 4], [0, 3, 6]]

[[1, 2, 3, 4], [2, 4, 6, 8], [3, 6, 9, 12]]

None of above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

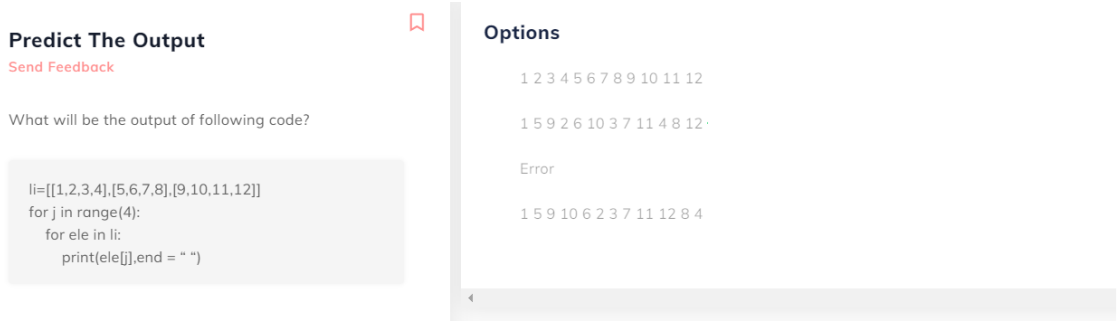
```
li = [[ i*j for j in range(4)] for i in range(3)]
print(li)
```

Options

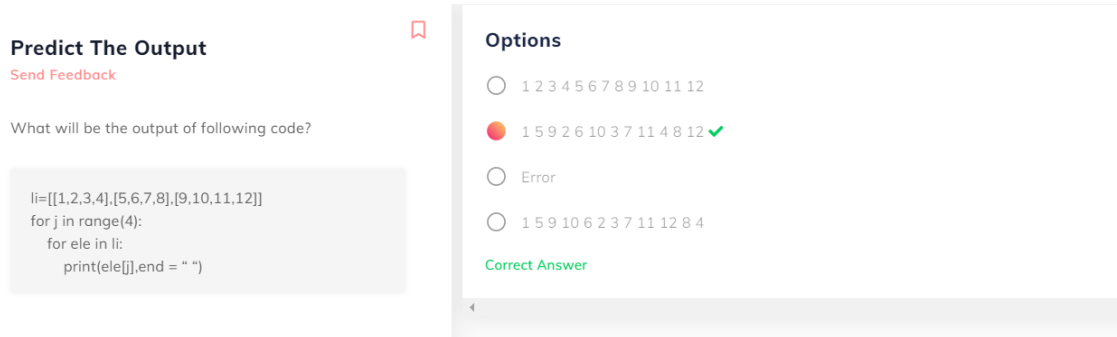
- [[0, 0, 0, 0], [0, 1, 2, 3], [0, 2, 4, 6]] ✔
- [[0, 0, 0], [0, 1, 2], [0, 2, 4], [0, 3, 6]]
- [[1, 2, 3, 4], [2, 4, 6, 8], [3, 6, 9, 12]]
- None of above

Correct Answer

Quiz



Solution



==========

Tuples:

A tuple is a collection which is ordered and unchangable(immutable). In python tuples are written with () round brackets.

NOTE: Python Tuple is used to store the sequence of immutable Python objects. The tuple is similar to lists since the value of the items stored in the list can be changed, whereas the tuple is immutable, and the value of the items stored in the tuple cannot be changed.

Create tuples: A variable is simply assigned to a set of comma-separated values within closed parentheses.

Example

In [173…
```
fruits=('apple','banana','orange','mango')
print(fruits)
```

```
('apple', 'banana', 'orange', 'mango')
```

len(): To determine how many item a tuple has, use the len().

Example

In [174…
```
fruits=('apple','banana','orange','mango')
print(len(fruits))
```

```
4
```

count(): Returns the number of times a specified value occurs in a tuple.

Example

In [175…
```
fruits=('apple','banana','orange','mango','mango')
```

```
print(fruits.count('mango'))
```

```
2
```

Indexing in a Tuple

- Indexing in a Tuple starts from index 0 .
- The highest index is the NumberOfElementsInTheTuple -1 .
- So a tuple having 10 elements would have indexing from 0-9 .
- This system is just like the one followed in Lists.

Negative Indexing

- Python language allows negative indexing for its tuple elements.
- The last element in the tuple has an index -1 , the second last element has index -2 ,and so on.

Element	1	15	13	18	19	23	4	5	2	3
Index	0	1	2	3	4	5	6	7	8	9
Negative indexing	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Example

In [176…
```
myTemp=( 1 , 15 , 13 , 18 , 19 , 23 , 4 , 5 , 2 , 3 ) #Number of element
print(myTemp[0])
print(myTemp[1])
print(myTemp[2])
print(myTemp[-1])
print(myTemp[-2])
```

```
1
15
13
3
2
```

tuple() constructor: It is also possible to use the tuple() constructor to make a tuple.

Example

In [177…
```
fruits=()
print(fruits)
print(type(fruits))
```

```
()
<class 'tuple'>
```

Slicing:

- We can slice a Tuple and then access any required range of elements. Slicing is doneby the means of a slicing operator which is ":".
- The basic format of slicing is:

Syntax:

In [ ]:
```
<Name of Tuple>[start index: end index]
```

Example:

In [178…
```
print(myTemp[1:4])
print(myTemp[7:])
```

```
(15, 13, 18)
(5, 2, 3)
```

What Changes Can Be Made To A Tuple

A tuple is immutable. This means that the objects or elements in a tuple cannot be changed once a tuple has been initialized. This is contrary to the case of lists - as a list is mutable, and hence the elements can be changed there.

- Python enables us to reassign the variables to different tuples.
- For example , the variable myTemp which has been assigned a tuple. To this variable some other tuple can be reassigned.

In [179…
```
myTemp = ([ 6 , 7 ], "Amity" , 2 , "boy" )
print(myTemp)
```

```
([6, 7], 'Amity', 2, 'boy')
```

- However, you cannot change any single element in the tuple.

In [180…
```
myTemp[ 2 ] = "Amity"
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-180-b7e008f83719> in <module>
----> 1 myTemp[ 2 ] = "Amity"

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

- Though, you can change items from any mutable object in the tuple.

In [181…
```
myTemp[ 0 ][ 1 ] = "Amity"
print(myTemp)
```

```
([6, 'Amity'], 'Amity', 2, 'boy')
```

- If you want to delete a tuple entirely, such an operation is possible.
- The keyword used is del.

In [182…
```
del (myTemp)
print(myTemp)
```

```
---------------------------------------------------------------------------
--
NameError                                 Traceback (most recent call las
t)
<ipython-input-182-afde73c2b15a> in <module>
      1 del (myTemp)
----> 2 print(myTemp)

NameError: name 'myTemp' is not defined
```

Looping through an entire element

We can use loops to iterate through a tuple : For example, if we want to print all the elements in the tuple. We can run the following loop.

Example

In [183…
```
myTemp=( 1 , 2 , 3 )
for t in myTemp:
    print(t)
```

```
1
2
3
```

==========

Set:

A Python set is the collection of the unordered items. Each element in the set must be unique, immutable, and the sets remove the duplicate elements. Sets are mutable which means we can modify it after its creation.

Unlike other collections in Python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together, or we can get the list of elements by looping through the set.

Creating a set: A set is created by using the set() function or placing all the elements within a pair of curly braces, separated by commas.

Example

In [184…
```
fruits={'apple','banana','cherry','mango'}
print(fruits)
```

```
{'mango', 'apple', 'banana', 'cherry'}
```

NOTE: The order of the elements has changed in the result.

Accessing Values in a Set: We cannot access individual values in a set as there is no specific order of elements in a set. We can only access all the elements together as shown. We can also get a list of individual elements by looping through the set.

Example

In [185…
```
Days=set([ "Mon" , "Tue" , "Wed" , "Thu" , "Fri" , "Sat" , "Sun" ])
for d in Days:
    print(d)
```

```
Sat
Sun
Wed
Fri
Thu
Tue
Mon
```

add(): To add one item to a set use the add() method.

Example

In [186…
```
fruits={'apple','banana','cherry','mango'}
print(fruits)
```

```
fruits.add('pineapple')
print(fruits)
```

```
{'mango', 'apple', 'banana', 'cherry'}
{'mango', 'pineapple', 'cherry', 'banana', 'apple'}
```

update(): To add more than one item to a set use the update() method.

Example

In [187…
```
fruitsname={'apple','banana','cherry','mango'}
print(fruits)

fruitsname.update('pineapple','guava')
print(fruits)
```

```
{'mango', 'pineapple', 'cherry', 'banana', 'apple'}
{'mango', 'pineapple', 'cherry', 'banana', 'apple'}
```

len(): To determine how many items a set has, use() the length method.

Example

In [188…
```
fruits={'apple','banana','cherry','mango'}
print(len(fruits))
```

```
4
```

remove(): To remove an item in a set use remove() method.

Example

In [189…
```
fruits={'apple','banana','cherry','mango'}
print(fruits)

fruits.remove('banana')
print(fruits)
```

```
{'mango', 'apple', 'banana', 'cherry'}
{'mango', 'apple', 'cherry'}
```

discard(): Discard is same as remove but if the to remove does ot exist, remove() will raise an error.

Example

In [190…
```
fruits={'apple','banana','cherry','mango'}
print(fruits)

fruits.discard('apple')
print(fruits)
```

```
{'mango', 'apple', 'banana', 'cherry'}
{'mango', 'banana', 'cherry'}
```

pop(): You can also use the pop(), method to remove an item, but this method will remove, the last item. Rememeber that sets are unordered, so you will not known what item that gets removed.

Example

In [191…
```
fruits={'apple','banana','cherry','mango'}
print(fruits)

fruits.pop()
print(fruits)
```

```
{'mango', 'apple', 'banana', 'cherry'}
{'apple', 'banana', 'cherry'}
```

clear(): The clear method empties the set.

Example

In [192…
```
fruits={'apple','banana','cherry','mango'}
print(fruits)

fruits.clear()
print(fruits)
```

```
{'mango', 'apple', 'banana', 'cherry'}
set()
```

del: The del keyword will delete the set completely.

Example

In [193…
```
fruits={'apple','banana','cherry','mango'}
print(fruits)

del fruits
print(fruits)
```

```
{'mango', 'apple', 'banana', 'cherry'}
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last
t)
<ipython-input-193-a35cf71ef81d> in <module>
      3
      4 del fruits
----> 5 print(fruits)

NameError: name 'fruits' is not defined
```

union(): Union of two given sets is the smallest set which contains all the elements of both the sets. Union of two given sets A and B is a set which consists of all the elements of A and all the elements of B such that no element is repeated.

Example

In [194…
```
fruits={'apple','banana','cherry','mango','carrot'}
print(fruits)

vegetables={'potato','onion','brinjal','carrot'}
print(vegetables)

print(fruits.union(vegetables))
```

```
{'mango', 'cherry', 'banana', 'carrot', 'apple'}
{'carrot', 'potato', 'onion', 'brinjal'}
{'mango', 'cherry', 'onion', 'banana', 'carrot', 'potato', 'apple', 'brin
jal'}
```

intersection(): Intersection of two given sets is the largest set which contains all the elements that are common to both the sets. Intersection of two given sets A and B is a set which consists of all the elements which are common to both A and B.

Example

In [195…
```
fruits={'apple','banana','cherry','mango','carrot'}
print(fruits)
```

```
vegetables={'potato','onion','brinjal','carrot'}
print(vegetables)

print(fruits.intersection(vegetables))
```

```
{'mango', 'cherry', 'banana', 'carrot', 'apple'}
{'carrot', 'potato', 'onion', 'brinjal'}
{'carrot'}
```

difference(): The difference between the two sets in Python is equal to the difference between the number of elements in two sets. The function difference() returns a set that is the difference between two sets. Let's try to find out what will be the difference between two sets A and B. Then (set A – set B) will be the elements present in set A but not in B and (set B – set A) will be the elements present in set B but not in set A.

Example

In [196…

```
fruits={'apple','banana','cherry','mango','carrot'}
print(fruits)

vegetables={'potato','onion','brinjal','carrot'}
print(vegetables)

print(fruits.difference(vegetables))

print(vegetables.difference(fruits))
```

```
{'mango', 'cherry', 'banana', 'carrot', 'apple'}
{'carrot', 'potato', 'onion', 'brinjal'}
{'banana', 'mango', 'apple', 'cherry'}
{'potato', 'onion', 'brinjal'}
```

isdisjoint(): Returns whether two sets have a intersection or not.

Example

In [197…

```
fruits={'apple','banana','cherry','mango','carrot'}
print(fruits)

vegetables={'potato','onion','brinjal','carrot'}
print(vegetables)

print(fruits.isdisjoint(vegetables))

fruits={'apple','banana','cherry','mango'}
print(fruits)

vegetables={'potato','onion','brinjal'}
print(vegetables)

print(fruits.isdisjoint(vegetables))
```

```
{'mango', 'cherry', 'banana', 'carrot', 'apple'}
{'carrot', 'potato', 'onion', 'brinjal'}
False
{'mango', 'apple', 'banana', 'cherry'}
{'potato', 'onion', 'brinjal'}
True
```

issubset(): Returns wheather another set contain this set or not.

Example

In [198…

```
fruits={'apple','banana','cherry','mango','carrot'}
```

```
print(fruits)

vegetables={'carrot'}
print(vegetables)

print(vegetables.issubset(fruits))

print(fruits.issubset(vegetables))
```

```
{'mango', 'cherry', 'banana', 'carrot', 'apple'}
{'carrot'}
True
False
```

issuperset(): Returns wheather this set contains another set or not.

Example

In [199…

```
fruits={'apple','banana','cherry','mango','carrot'}
print(fruits)

vegetables={'carrot'}
print(vegetables)

print(vegetables.issuperset(fruits))

print(fruits.issuperset(vegetables))
```

```
{'mango', 'cherry', 'banana', 'carrot', 'apple'}
{'carrot'}
False
True
```

==========

Dictionary:

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key value is provided in the dictionary to make it more optimized.

- A Dictionary is a Python's implementation of a data structure that is more generally known as an associative array.
- A dictionary is an unordered collection of key-value pairs.
- Indexing in a dictionary is done using these "keys".
- Each pair maps the key to its value.
- Literals of the type dictionary are enclosed within curly brackets.
  Within these brackets, each entry is written as a key followed by a colon " : ", which is further followed by a value. This is the value that corresponds to the given key.
- These keys must be unique and cannot be any immutable data type. Eg- string, integer, tuples, etc. They are always mutable.
- The values need not be unique. They can be repetitive and can be of any data type (Both mutable and immutable)
- Dictionaries are mutable, which means that the key-value pairs can be changed.

How to create a dictionary: Simply put pairs of key-value within curly brackets. Keys are

separated from their corresponding values by a colon. Different key-value pairs are separated from each other by commas. Assign this to a variable.

Example

In [200…
```
aliens = {'color': 'green', 'points': 5}
print(aliens['color'])
print(aliens['points'])
```

```
green
5
```

A dictionary in Python is a collection of key-value pairs. Each key is connected to a value, and you can use a key to access the value associated with that key. A key's value can be a number, a string, a list, or even another dictionary. In fact, you can use any object that you can create in Python as a value in a dictionary. In Python, a dictionary is wrapped in braces, {}, with a series of keyvalue pairs inside the braces, as shown in the earlier example:

A key-value pair is a set of values associated with each other. When you provide a key, Python returns the value associated with that key. Every key is connected to its value by a colon, and individual key-value pairs are separated by commas. You can store as many key-value pairs as you want in a dictionary.

Accessing Values in a Dictionary: To get the value associated with a key, give the name of the dictionary and then place the key inside a set of square brackets, as shown here:

In [201…
```
alien_0 = {'color': 'green'}
print(alien_0['color'])
```

```
green
```

Adding New Key-Value Pairs: Dictionaries are dynamic structures, and you can add new key-value pairs to a dictionary at any time. For example, to add a new key-value pair, you would give the name of the dictionary followed by the new key in square brackets along with the new value.

Example

In [202…
```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)

alien_0['x_position'] = 0
alien_0['y_position'] = 25

print(alien_0)
```

```
{'color': 'green', 'points': 5}
{'color': 'green', 'points': 5, 'x_position': 0, 'y_position': 25}
```

Modifying Values in a Dictionary: To modify a value in a dictionary, give the name of the dictionary with the key in square brackets and then the new value you want associated with that key

Example

In [203…
```
alien_0 = {'color': 'green'}
print("The alien is " + alien_0['color'] + ".")
alien_0['color'] = 'yellow'
print("The alien is now " + alien_0['color'] + ".")
```

```
The alien is green.
The alien is now yellow.
```

Removing Key-Value Pairs: When you no longer need a piece of information that's stored in a dictionary, you can use the del statement to completely remove a key-value pair. All del needs is the name of the dictionary and the key that you want to remove.

Example

In [204…
```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)

del alien_0['points']
print(alien_0)
```

```
{'color': 'green', 'points': 5}
{'color': 'green'}
```

Looping through all the dictionary:

Example

In [205…
```
user_0 = {
'username': 'efermi',
'first': 'enrico',
'last': 'fermi',
}

for key, value in user_0.items():
    print("\nKey: " + key)
    print("Value: " + value)
```

```
Key: username
Value: efermi

Key: first
Value: enrico

Key: last
Value: fermi
```

len(): To determines how many items(key-value pairs) in a dictionary, use the len() function.

Example

In [206…
```
user_0 = {
'username': 'efermi',
'first': 'enrico',
'last': 'fermi',
}

print(len(user_0))
```

```
3
```

pop(): The pop() method to remove items from a dictionary.

Example

In [207…
```
user_0 = {
'username': 'efermi',
'first': 'enrico',
'last': 'fermi',
}
```

```
user_0.pop("first")
print(user_0)
```

```
{'username': 'efermi', 'last': 'fermi'}
```

popitem(): The popitem() method removes the last inserted item.

Example

In [208…
```
user_0 = {
'username': 'efermi',
'first': 'enrico',
'last': 'fermi',
}

user_0.popitem()
print(user_0)
```

```
{'username': 'efermi', 'first': 'enrico'}
```

del: The del keyword removes the item with the specified key name.

Example

In [209…
```
user_0 = {
'username': 'efermi',
'first': 'enrico',
'last': 'fermi',
}

del user_0['first']
print(user_0)
```

```
{'username': 'efermi', 'last': 'fermi'}
```

copy(): The clear() method empties the dictionary.

Example

In [210…
```
user_0 = {
'username': 'efermi',
'first': 'enrico',
'last': 'fermi',
}

print(user_0)

user_1 = user_0.copy()
print(user_1)
```

```
{'username': 'efermi', 'first': 'enrico', 'last': 'fermi'}
{'username': 'efermi', 'first': 'enrico', 'last': 'fermi'}
```

update(): The update() method inserts the specified item to the dictionary.
The specified item can be a dictionary, or an iterable objects.

Example

In [211…
```
user_0 = {
'username': 'efermi',
'first': 'enrico',
'last': 'fermi',
}

user_0.update({'x_coordinate' : 10,'y_coordinate' : 20})
print(user_0)
```

```
{'username': 'efermi', 'first': 'enrico', 'last': 'fermi', 'x_coordinat
e': 10, 'y_coordinate': 20}
```

Mini project Apni Dictionary

Create a simple dictionary when a user search for a particular word if a particular word present in the dictionary. It will give you meaning of that particular.[important just add 5 words in dictionary]

In [213…

```
dictionary={'abandoned':'left completely and no longer used or wanted',
            'adorable':'very attractive and easy to feel love for',
            'bipolar':'having or relating to two poles or extremities.',
            'consistency':'the quality of always having the same standar
            'diversity':'the wide variety of something'}

word=input("Enter the word you want to search: ")

if word in dictionary:
    print("The meaning of {} is {}".format(word,dictionary[word]))
else:
    print("This word is not present in the dictionary")
```

```
Enter the word you want to search: abandoned
The meaning of abandoned is left completely and no longer used or wanted
```

==========

Object Oriented Programming(Oops):

Introduction to Oops:

Object-oriented programming is a programming paradigm that provides a means of structuring programs so that properties and behaviors are bundled into individual objects. For instance, an object could represent a person with properties like a name, age, and address and behaviors such as walking, talking, breathing, and running. Or it could represent an email with properties like a recipient list, subject, and body and behaviors like adding attachments and sending.
Put another way, object-oriented programming is an approach for modeling concrete, real-world things, like cars, as well as relations between things, like companies and employees, students and teachers, and so on. OOP models real-world entities as software objects that have some data associated with them and can perform certain functions.

Class:

A class is a blueprint that defines the variables and the methods (Characteristics) common to all objects of a certain kind. Example: If Car is a class, then Maruti 800 is an object of the Car class. All cars share similar features like 4 wheels, 1 steering wheel, windows, breaks etc. Maruti 800 (The Car object) has all these features.

Defining a Class

All class definitions start with the class keyword, which is followed by the name of the class and a colon(:). Any code that is indented below the class definition is considered part of the class's body.

Here is an example of a Car class:

Example

In [215…
```
class Car:
    pass
```

The body of the Car class consists of a single statement: the pass keyword. As we have discussed earlier, pass is often used as a placeholder indicating where code will eventually go. It allows you to run this code without Python throwing an error.

Note: Python class names are written in Capitalized Words notation by convention. For example, a class for a specific model of Car like the Bugatti Veyron would be written as BugattiVeyron. The first letter is capitalized. This is just a good programming practice.

Object:

The object is an entity that has a state and a behavior associated with it. It may be any real-world object like the mouse, keyboard, chair, table, pen, etc.

Integers, strings, floating-point numbers, even arrays and dictionaries, are all objects. More specifically, any single integer or any single string is an object. The number 12 is an object, the string "Hello, world" is an object, a list is an object that can hold other objects, and so on. You've been using objects all along and may not even realize it.

Example

In [216…
```
c1 = Car()
```

Here, c1 is an object of class car.

In [217…
```
print(type(c1))     #Let's check the type of c1:
print(id(c1))       #Where c1 object is stored in the memory
```

```
<class '__main__.Car'>
2310379413408
```

NOTE:

Till now we have learnt so many concept of python and in the starting. I told you all the things in python is object like data type int, string, list, tuple, set and dictionary. Let's have a look into it

In [218…
```
X=15

print(type(X))
print("**************")

X="Sachin"

print(type(X))
print("**************")
X=[0,1,2,3,4,5]

print(type(X))
print("**************")
X=(0,1,2,3,4,5)

print(type(X))
print("**************")
X={0,1,2,3}
```

```
print(type(X))
print("**************")
X={1:"Sachin", 2:"Ranjit", 3:"Kundan"}

print(type(X))
```

```
<class 'int'>
**************
<class 'str'>
**************
<class 'list'>
**************
<class 'tuple'>
**************
<class 'set'>
**************
<class 'dict'>
```

As you can see from above code all the data types have inbuilt classes in python we just create a instance of that class. We are using objects all along and may not even realize it.

Classes vs Objects (Or Instances):

Classes are used to create user-defined data structures. Classes define functions called methods, which identify the behaviors and actions that an object created from the class can perform with its data.

A class is a blueprint for how something should be defined. It doesn't contain any data. The Car class specifies that a name and a top-speed are necessary for defining a Car, but it doesn't contain the name or top-speed of any specific Car.

While the class is the blueprint, an instance is an object that is built from a class and contains real data. An instance of the Car class is not a blueprint anymore. It's an actual car with a name, like Creta, and with a top speed of 200 Km/Hr.

Put another way, a class is like a form or questionnaire. An instance is like a form that has been filled out with information. Just like many people can fill out the same form with their unique information, many instances can be created from a single class.

init method

The init method is a constructor. Constructors are a concept from the object-oriented programming. A class can have one and only one constructor. If init is defined within a class, it is automatically invoked when we create a new class instance.

- Constructors are generally used for instantiating an object.
- The task of a constructor is to initialize(assign values) to the data members of the class when an object of the class is created.
- In Python, the .__init__() method is called the constructor and is always called when an object is created.

Note: Names that have leading and trailing double underscores are reserved for special use like the **init** method for object constructors. These methods are known as dunder methods.

Syntax:

In [ ]:
```
def __init__(self):
    # body of the constructor
```

Types of constructor: There are two types of constructor.

1. **Default Constructor:** The default constructor is a simple constructor that doesn't accept any arguments. Its definition has only one argument which is a reference to the instance being constructed known as self.

Example

In [219…
```
class Car:
    def __init__(self):    #__init__ method default constructor
        print("This is a default constructor")


c1=Car()
c2=Car()
```

```
This is a default constructor
This is a default constructor
```

2. Parameterized Constructor: A constructor with parameters is known as a parameterized constructor. The parameterized constructor takes its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.

Example

In [220…
```
class Car:
    def __init__(self,brand,model,engine_displacement):    #__init__ meth
        self.brand=brand
        self.model=model
        self.engine_displacement=engine_displacement


c1=Car("BMW","BMW 3",6523)
c2=Car("Mercedes","Benz C class",6580)

print("c1 object details: ")
print(c1.brand)
print(c1.model)
print(c1.engine_displacement)
print()
print("c2 object details: ")
print(c2.brand)
print(c2.model)
print(c2.engine_displacement)
```

```
c1 object details:
BMW
BMW 3
6523

c2 object details:
Mercedes
Benz C class
6580
```

NOTE: The properties that all Car objects must have been defined in .**init**(). Every time a new Car object is created, .**init**() sets the initial state of the object by assigning the values

of the object's properties. That is, .**init**() initializes each new instance of the class.

When a new class instance is created, the instance is automatically passed to the self parameter in .**init**() so that new attributes can be defined on the object.

Self Parameter

You may have noticed that our init method had another argument besides brand,model and engine_displacement: self. The self argument represents a particular instance of the class and allows us to access its attributes and methods. In the example with init, we basically create attributes for the particular instance and assign the values of method arguments to them. It is important to use the self parameter inside the method if we want to save the values of the instance for the later use.

Most of the time we also need to write the self parameter in other methods because when the method is called the first argument that is passed to the method is the object itself. Let's add a method to our Car class and see how it works. The syntax of the methods is not of importance at the moment, just pay attention to the use of the self:

- The self parameter is a reference to the current instance of the class and is used to access variables that belong to the class.
- It does not have to be named self, you can call it whatever you like, but it has to be the first parameter of any function in the class.
- You can give .__init__() any number of parameters, but the first parameter will always be a variable called self.

Example

In [221…

```
class Car:
    def __init__(self,brand,model,engine_displacement):
        self.brand=brand
        self.model=model
        self.engine_displacement=engine_displacement

    def get_info(self):
        print("The brand of the car is {0} and model number is {1} with

c1=Car("BMW","BMW 3",6523)
c2=Car("Mercedes","Benz C class",6580)

print("Object c1 details: ")
print(c1.brand)
print(c1.model)
print(c1.engine_displacement)
print(c1.get_info())
print()
print("Object c2 details: ")
print(c2.brand)
print(c2.model)
print(c2.engine_displacement)
print(c2.get_info())
```

```
Object c1 details:
BMW
```

```
BMW 3
6523
The brand of the car is BMW and model number is BMW 3 with engine displac
ement 6523
None

Object c2 details:
Mercedes
Benz C class
6580
The brand of the car is Mercedes and model number is Benz C class with en
gine displacement 6580
None
```

Note: The .**init**() method's signature is indented four spaces. The body of the method is indented by eight spaces. This indentation is vitally important. It tells Python that the .**init**() method belongs to the Car class.
In the body of .**init**(), three statements are using the self variable:

1. self.brand = brand creates an attribute called brand and assigns to it the value of the brand parameter.
2. self.model = model creates an attribute called model and assigns to it the value of the model parameter.
3. self.engine_displacement = engine_displacement creates an attribute called engine_displacement and assigns to it the value of the engine_displacement parameter.

Types of variables

In oops we have two types of variable.

1. **Instance variable:**Variable created in .__init__() are called instance variable. An instance variable's value is specific to a particular instance of the class. All Car objects have a brand,model and a engine_displacement, but the values for the brand,model and engine_displacement attributes will vary depending on the Car instance. Different objects of the Car class will have different brand,model and engine_displacement.
2. **Class/Static variable:**class/Static variable are attributes that have the same value for all class instances. You can define a class attribute by assigning a value to a variable name outside of .__init__().

Example

In [222…

```
class Car:
    color="Black"   #Class variable
    def __init__(self,brand,model,engine_displacement):
        self.brand=brand    #Instance variable
        self.model=model
        self.engine_displacement=engine_displacement

    def get_info(self):
        print(f"The brand of the car is {self.brand} and model number is


c1=Car("BMW","BMW 3",6523)
c2=Car("Mercedes","Benz C class",6580)
```

```
print("Object c1 details: ")
print(c1.brand)
print(c1.model)
print(c1.engine_displacement)
print(c1.color)
print(c1.get_info())
print()
print("Object c2 details: ")
print(c2.brand)
print(c2.model)
print(c2.engine_displacement)
print(c1.color)
print(c2.get_info())
```

```
Object c1 details:
BMW
BMW 3
6523
Black
The brand of the car is BMW and model number is BMW 3 with engine displac
ement 6523 and in color Black
None

Object c2 details:
Mercedes
Benz C class
6580
Black
The brand of the car is Mercedes and model number is Benz C class with en
gine displacement 6580 and in color Black
None
```

NOTE:

- Class attributes are defined directly beneath the first line of the class name and are indented by four spaces.
- They must always be assigned an initial value.
- When an instance of the class is created, the class attributes are automatically created and assigned to their initial values.

Types of method

In oops we have three types of methods:

1. Instance method: Instance attributes are those attributes that are not shared by objects. Every object has its own copy of the instance attribute.
If we are using self as a function parameter or in front of a variable, that is nothing but the calling instance itself.

As we are working with instance variables we use self keyword.
Note: Instance variables are used with instance methods.

Example

In [223…

```
class Car:
    color="Black"    #Class variable
    def __init__(self,brand,model,engine_displacement):
        self.brand=brand    #Instance variable
        self.model=model
```

```
        self.engine_displacement=engine_displacement

    def get_info(self): #instance method
        print(f"The brand of the car is {self.brand} and model number is

c1=Car("BMW","BMW 3",6523)
print(c1.get_info())
```

```
The brand of the car is BMW and model number is BMW 3 with engine displac
ement 6523 and in color Black
None
```

2. Static method: Static methods, is shared by objects. Every object has to share the method. So, they are not dependent on the state of the object.

A static method can be called without an object for that class, using the class name directly. If you want to do something extra with a class we use static methods.

Example

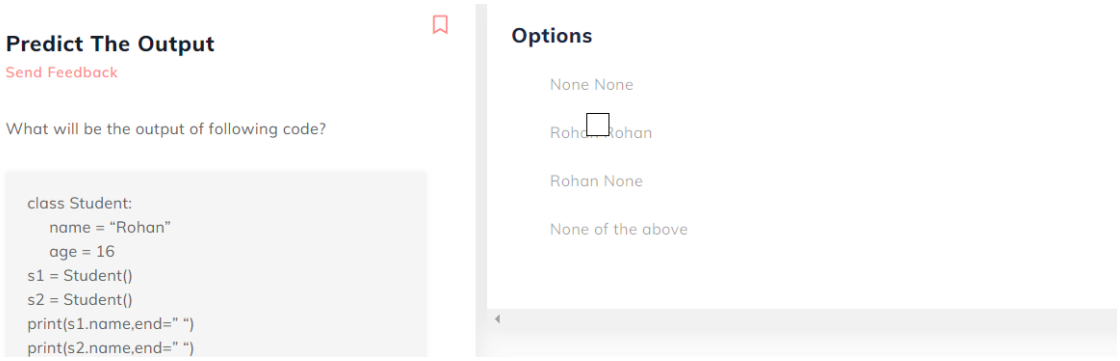
In [224…

```
class Car:
    color="Black"   #Class variable
    def __init__(self,brand,model,engine_displacement):
        self.brand=brand    #Instance variable
        self.model=model
        self.engine_displacement=engine_displacement

    @staticmethod
    def get_color_of_car():
        print(f"The color of the car is {Car.color}")


c1=Car("BMW","BMW 3",6523)
print(c1.get_color_of_car())
print(Car.get_color_of_car())
```

```
The color of the car is Black
None
The color of the car is Black
None
```

As you can see we can call the method using object or via class as well. Because this method belongs to class. So, you can call the method via class.

Quiz



Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:
    name = "Rohan"
    age = 16
s1 = Student()
s2 = Student()
print(s1.name,end=" ")
print(s2.name,end=" ")
```

Options

- None None
- Rohan Rohan ✔
- Rohan None
- None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:
    pp = 50

s1 = Student()
s1.pp= 58
s2 = Student()
s2.pp = 60
print(s1.pp)
```

Options

- 50
- 58
- 60
- None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:
    pp = 50

s1 = Student()
s1.pp= 58
s2 = Student()
s2.pp = 60
print(s1.pp)
```

Options

- 50
- 58 ✔
- 60
- None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:
    name = "Parikh"
    def store_details(self):
        self.age = 60
    def print_details(self):
        print(self.name, end=" ")
        print(self.age)
s = Student()
s.store_details()
s.print_details()
```

Options

Error

Parikh 60

Parikh None

None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:
    name = "Parikh"
    def store_details(self):
        self.age = 60
    def print_details(self):
        print(self.name, end=" ")
        print(self.age)
s = Student()
s.store_details()
s.print_details()
```

Options

- Error
- Parikh 60 ✔
- Parikh None
- None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:
    name = "Parikh"
    def store_details(self):
        self.age = 60
    def print_age(self):
        print(self.age)
s = Student()
s.store_details()
s1 = Student()
s1.print_age()
```

Options

Error

60

None

Parikh 60

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:
    name = "Parikh"
    def store_details(self):
        self.age = 60
    def print_age(self):
        print(self.age)
s = Student()
s.store_details()
s1 = Student()
s1.print_age()
```

Options

- Error ✔
- 60
- None
- Parikh 60

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:

    def __init__(self,name,age):
        self.name = "Rohan"
        self.age = 20

    def print_student_details():
        print(self.name, end= " ")
        print(self.age)


s = Student()
s.print_student_details()
```

Options

- Rohan 20
- Error
- Rohan
- None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:

    def __init__(self,name,age):
        self.name = "Rohan"
        self.age = 20

    def print_student_details():
        print(self.name, end= " ")
        print(self.age)


s = Student()
s.print_student_details()
```

Options

- Rohan 20
- Error ✔
- Rohan
- None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def print_student_details(self):
        print(self.name, end= " ")
        print(self.age)

s = Student("Rohan",60)
s.print_student_details()
```

Options

Error

Rohan 60

Rohan

None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def print_student_details(self):
        print(self.name, end= " ")
        print(self.age)

s = Student("Rohan",60)
s.print_student_details()
```

Options

Error

Rohan 60 ✔

Rohan

None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def print_student_details(self):
        print(self.name, end= " ")
        print(self.age)

s = Student("Rohan",60)
s.print_student_details()
```

Options

Error

Rohan 60

Rohan

None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class Student:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def print_student_details(self):
        print(self.name, end= " ")
        print(self.age)

s = Student("Rohan",60)
s.print_student_details()
```

Options

- Error
- Rohan 60 ✔
- Rohan
- None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of this code?

```
class Student:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def print_student_details():
        print(self.name, end= " ")
        print(self.age)

    @staticmethod
    def isTeen(age):
        return age>16

a = Student.isTeen(18)
print(a)
```

Options

- True
- False
- Error
- None of the above

Solution

Predict The Output

Send Feedback

What will be the output of this code?

```
class Student:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def print_student_details():
        print(self.name, end= " ")
        print(self.age)

    @staticmethod
    def isTeen(age):
        return age>16

a = Student.isTeen(18)
print(a)
```

Options

- True ✔
- False
- Error
- None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of this code?

```
class Student:
    def __init__(self,name,age):
        self.__name = name
        self.age = age
    def print_student_details():
        print(self.__name, end= " ")
        print(self.age)

s = Student("Rohan",20)
print(s.name)
```

Options

Rohan

None

Error

None of the above

Solution

Predict The Output

Send Feedback

What will be the output of this code?

```
class Student:
    def __init__(self,name,age):
        self.__name = name
        self.age = age
    def print_student_details():
        print(self.__name, end= " ")
        print(self.age)

s = Student("Rohan",20)
print(s.name)
```

Options

- ○ Rohan
- ○ None
- ● Error ✔
- ○ None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of this code?

```
class Student:
    def __init__(self,name,age):
        self.__name = name
        self.age = age
    def print_student_details():
        print(self.__name, end= " ")
        print(self.age)

s = Student("Rohan",20)
print(s.name)
```

Options

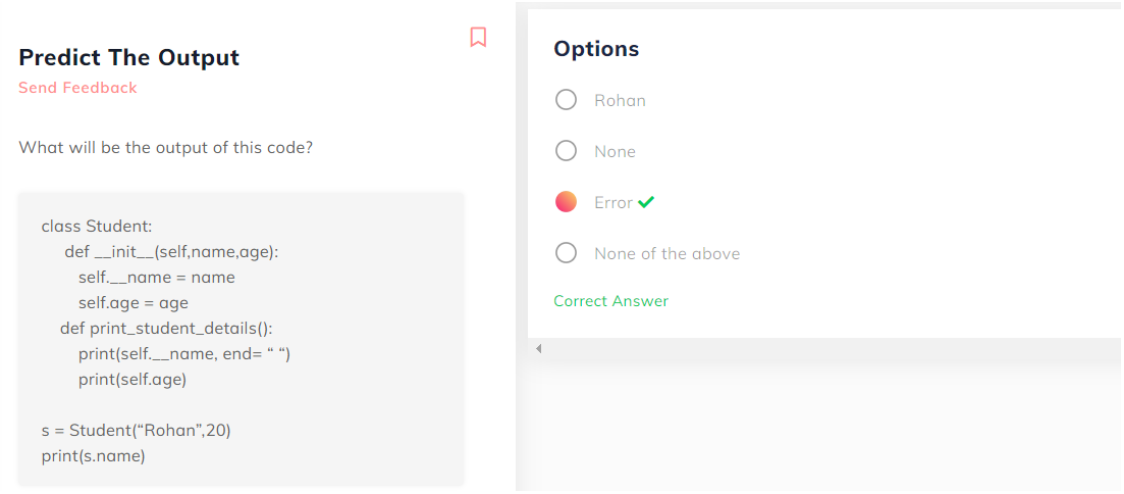
Rohan

None

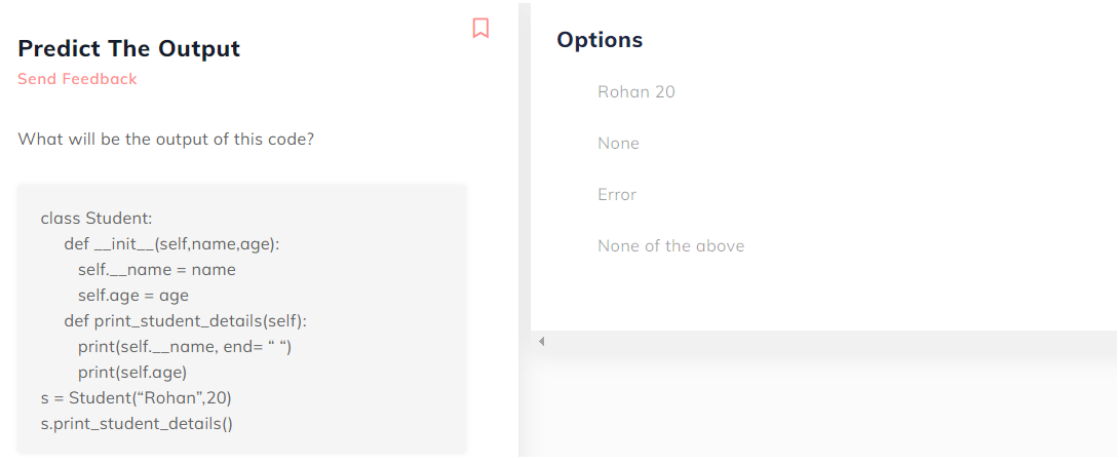
Error

None of the above

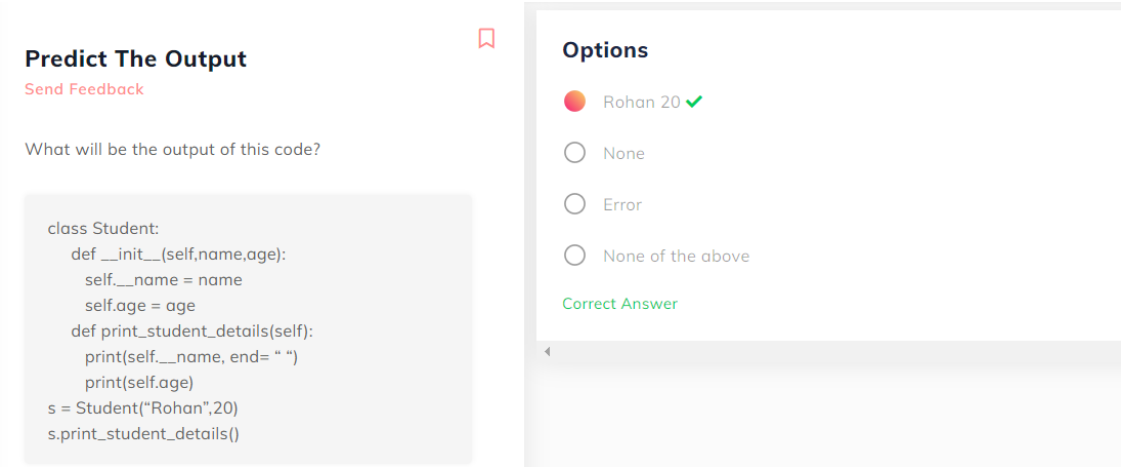
Solution



Quiz



Solution



==========

Inheritance

Inheritance is a mechanism that allows classes to inherit methods or properties from other classes. Or, in other words, inheritance is a mechanism of deriving new classes from existing ones.

The purpose of inheritance is to reuse existing code. Often, objects of one class may resemble objects of another class, so instead of rewriting the same methods and attributes, we can make it so that a class inherits those methods and attributes from another class.

When we talk about inheritance, the terminology resembles biological inheritance: we have child classes (or subclasses, derived classes) that inherit methods or variables from parent classes (or base classes, superclasses). Child classes can also redefine methods of the parent class if necessary.

Inheritance is very easy to implement in your programs. Any class can be a parent class, so all we need to do is to write in the definition of the child class the name of the parent class in parentheses after the child class:

Syntax

In [225…
```
class Parent:
    #variable and attribute of parent class
    pass

class Child(Parent):
    #variable and attribute of child class
    pass
```

NOTE:

The definition of the parent class should precede the definition of the child class, otherwise, you'll get a NameError! If a class has several subclasses, its definition should precede them all. The "sibling" classes can be defined in any order.

Example

In [226…
```
#Definition of parent class
class Person:
    def __init__(self,Fname,Lname):
        self.firstname=Fname
        self.lastname=Lname
    def show(self):
        print("The first name is {0} and last name is {1}".format(self.f

p1=Person("Sachin","Kapoor")
p1.show()
p2=Person("Ranjeet","Kumar")
p2.show()
#Definition of child class
class Student(Person):
    pass

s1=Student("Abhishek","Jaiswal")
s1.show()
s2=Student("Rishav","Goyal")
s2.show()
```

```
The first name is Sachin and last name is Kapoor
The first name is Ranjeet and last name is Kumar
The first name is Abhishek and last name is Jaiswal
The first name is Rishav and last name is Goyal
```

NOTE: As you can see Person class is parent class and Student is child class. Student class inherit all the variable and method from parent class. Let's check the how many variable Child class have. To chech which object have how many variable you can use **dict** with object. So, you will get the dictionary of variable object have

In [227…
```
s1.__dict__
```

Out[227…
```
{'firstname': 'Abhishek', 'lastname': 'Jaiswal'}
```

s1 have firstname and lastname as a varible because it inherit the parent class and parent class have firstname and lastname as variable. So student all the variable and method from parent class.

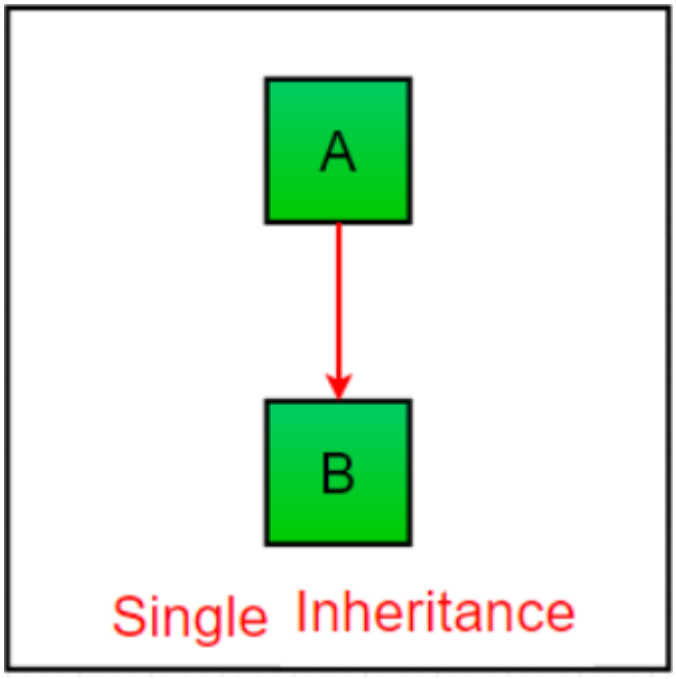
Types of inheritance

Inheritance is defined as the capability of one class to derive or inherit the properties from some other class and use it whenever needed. Inheritance provides the following properties:

- It represents real-world relationships well.
- It provides reusability of code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Types of Inheritance depends upon the number of child and parent classes involved. There are four types of inheritance in Python:

Single Inheritance: Single inheritance enables a derived class to inherit properties from a single parent class, thus enabling code reusability and the addition of new features to existing code.



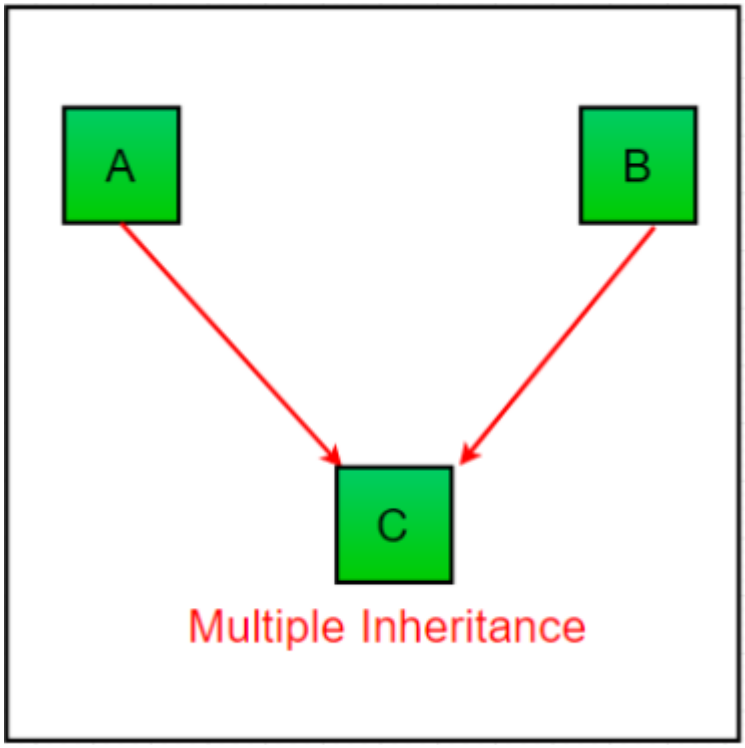
Example

```
# Python program to demonstrate
```

In [228…
```python
# single inheritance


# Base class
class Parent:
     def func1(self):
          print("This function is in parent class.")

# Derived class
class Child(Parent):
     def func2(self):
          print("This function is in child class.")

# Driver's code
object = Child()
object.func1()
object.func2()
```

```
This function is in parent class.
This function is in child class.
```

Multiple Inheritance: When a class can be derived from more than one base class this type of inheritance is called multiple inheritance. In multiple inheritance, all the features of the base classes are inherited into the derived class.



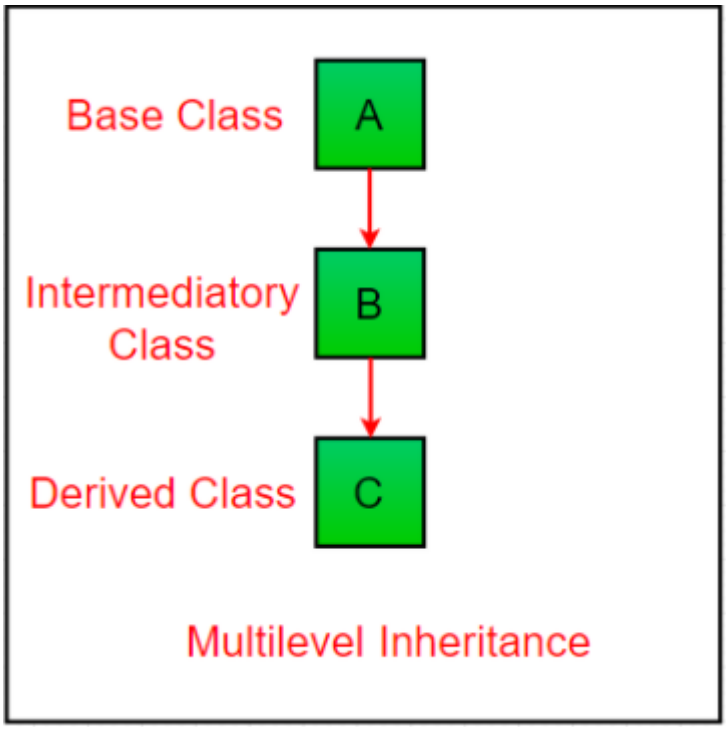
Example

In [229…
```python
# Python program to demonstrate
# multiple inheritance


# Base class1
class Mother:
    mothername = ""
    def mother(self):
        print(self.mothername)

# Base class2
class Father:
    fathername = ""
```

```
    def father(self):
        print(self.fathername)

# Derived class
class Son(Mother, Father):
    def parents(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)

# Driver's code
s1 = Son()
s1.fathername = "RAM"
s1.mothername = "SITA"
s1.parents()
```

```
Father : RAM
Mother : SITA
```

Multilevel Inheritance: In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class. This is similar to a relationship representing a child and grandfather.



Example

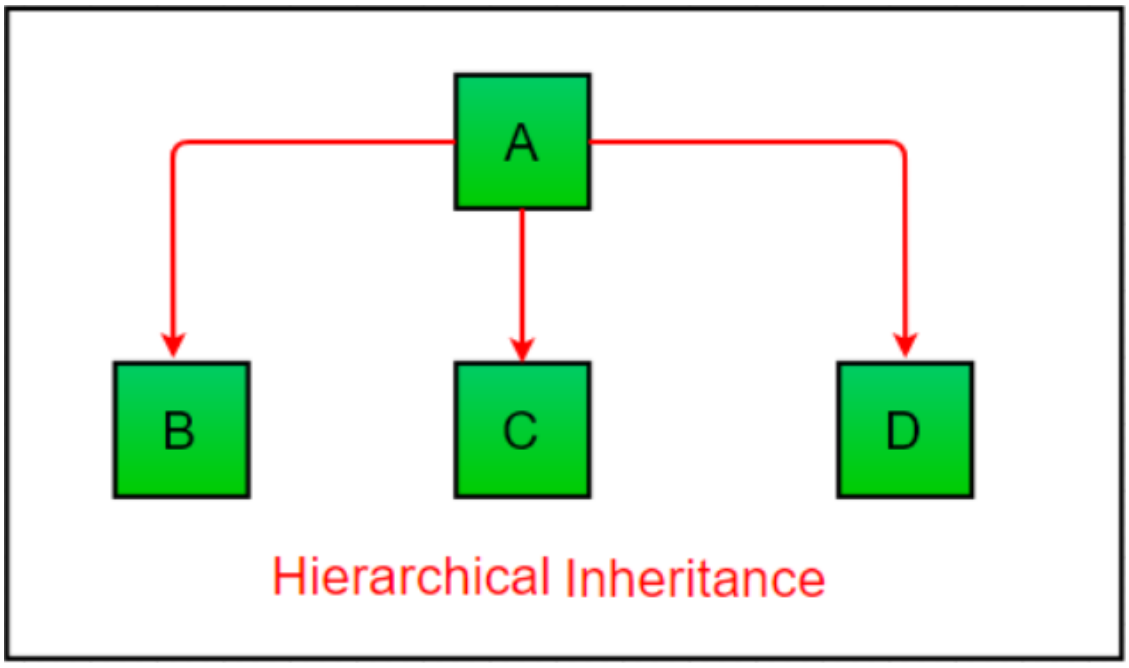
In [230…

```
# Python program to demonstrate
# multilevel inheritance

# Base class
class Grandfather:

    def __init__(self, grandfathername):
        self.grandfathername = grandfathername

# Intermediate class
class Father(Grandfather):
    def __init__(self, fathername, grandfathername):
        self.fathername = fathername

        # invoking constructor of Grandfather class
        Grandfather.__init__(self, grandfathername)
```

```
# Derived class
class Son(Father):
    def __init__(self,sonname, fathername, grandfathername):
        self.sonname = sonname

        # invoking constructor of Father class
        Father.__init__(self, fathername, grandfathername)

    def print_name(self):
        print('Grandfather name :', self.grandfathername)
        print("Father name :", self.fathername)
        print("Son name :", self.sonname)

#  Driver code
s1 = Son('Prince', 'Rampal', 'Lal mani')
print(s1.grandfathername)
s1.print_name()
```

```
Lal mani
Grandfather name : Lal mani
Father name : Rampal
Son name : Prince
```

Hierarchical Inheritance: When more than one derived classes are created from a single base this type of inheritance is called hierarchical inheritance. In this program, we have a parent (base) class and two child (derived) classes.



Example

In [231…

```
# Python program to demonstrate
# Hierarchical inheritance


# Base class
class Parent:
      def func1(self):
          print("This function is in parent class.")

# Derived class1
class Child1(Parent):
      def func2(self):
```

```
        print("This function is in child 1.")

# Derivied class2
class Child2(Parent):
      def func3(self):
          print("This function is in child 2.")

# Driver's code
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

```
This function is in parent class.
This function is in child 1.
This function is in parent class.
This function is in child 2.
```

Access modifiers

A Class in Python has three types of access modifiers –

- Public Access Modifier
- Protected Access Modifier
- Private Access Modifier

Public Access Modifier: The members of a class that are declared public are easily accessible from any part of the program. All data members and member functions of a class are public by default.

Example

In [232…

```
# program to illustrate public access modifier in a class
class Person:
    # constructor
    def __init__(self,Fname,Lname):
        self.firstname=Fname
        self.lastname=Lname

    # public memeber function
    def show(self):
        print("The first name is {0} and last name is {1}".format(self.f

# creating object of the class
p1=Person("Sachin","kapoor")

# accessing public data member
print(p1.firstname)
print(p1.lastname)
```

```
Sachin
kapoor
```

and in public access modifier if we want to change the variable there is no restriction in changing the variable.

In [233…

```
p1.firstname="Ranjeet"
p1.lastname="Kumar"
print(p1.firstname)
print(p1.lastname)
```

```
Ranjeet
Kumar
```

Example: During inheritance

In [234…
```
class Parent:

    def __init__(self,firstname,lastname):
        self.firstname=firstname
        self.lastname=lastname

class Child(Parent):

    def __init__(self,firstname,lastname,age):
        super().__init__(firstname,lastname)
        self.age=age

    def showdetails(self):
        print(f"My {self.firstname} {self.lastname} and age is {self.age


c1=Child("Sachin","Kapoor",26)
c1.showdetails()
```

```
My Sachin Kapoor and age is 26
```

Protected Access Modifier: The members of a class that are declared protected are only accessible to a class derived from it. Data members of a class are declared protected by adding a single underscore '_' symbol before the data member of that class.

NOTE: In python there is no such concept of protected you can use protected member outside the class in inherited class as well. Python thinks that the programmer is sensible and if any programmer see any variable with single underscore(_). So he/she should get that this is protected member and don't try to change outside the class

Example

In [235…
```
# program to illustrate public access modifier in a class
class Person:
    # constructor
    def __init__(self,Fname,Lname):
        self._firstname=Fname
        self._lastname=Lname

    # public memeber function
    def show(self):
        print("The first name is {0} and last name is {1}".format(self._

# creating object of the class
p1=Person("Sachin","kapoor")

# accessing public data member
print(p1._firstname)
print(p1._lastname)
```

```
Sachin
kapoor
```

Example: During inheritance

In [236…
```
class Person:
    _name=None
    _age=None
```

```
    def __init__(self,name,age):
        self._name=name
        self._age=age

    def display_nameandage(Self):
        print(f"My name is {self._name} and age is {self._age}")

class child(Person):
    def __init__(self,name,age):
        super().__init__(name,age)

    def displaydetails(self):
        print(f"Name is {self._name}")

c1=child("Sachin kapoor","23")
c1.displaydetails()
```

```
Name is Sachin kapoor
```

Private Access Modifier: The members of a class that are declared private are accessible within the class only, private access modifier is the most secure access modifier. Data members of a class are declared private by adding a double underscore '__' symbol before the data member of that class.

Example

In [237…

```
# program to illustrate private access modifier in a class
class Person:
    # constructor
    def __init__(self,Fname,Lname,age):
        self.firstname=Fname
        self.lastname=Lname
        self.__age=age        #private access modifier

    # public memeber function
    def show(self):
        print("The first name is {0} and last name is {1} and age is {2}

# creating object of the class
p1=Person("Sachin","kapoor",23)

# accessing public data member
print(p1.firstname)
print(p1.lastname)
```

```
Sachin
kapoor
```

as you can see in above code we can access public variable like firstname and lastname. Let's see can we access private variable age outside class or not?

In [238…

```
# accessing private data member
print(p1.__age)
```

```
---------------------------------------------------------------------------
--
AttributeError                            Traceback (most recent call las
t)
<ipython-input-238-de11d914012a> in <module>
      1 # accessing private data member
----> 2 print(p1.__age)

AttributeError: 'Person' object has no attribute '__age'
```

as you can see we can't change private data member outside the class.

In [239…
```
print(p1.show())
```

```
The first name is Sachin and last name is kapoor and age is 23
None
```

Example: During inheritnce

In [240…
```
class Person:

    def __init__(self,firstname,lastname):
        self.__firstname=firstname
        self.__lastname=lastname

    def display_nameandage(self):
        print(f"My name is {self.__firstname} and age is {self.__lastnam

class child(Person):

    def __init__(self,firstname,lastname,age):
        super().__init__(firstname,lastname)
        self.age=age

    def displaydetails(self):
        super().display_nameandage()
        print(f"Age is {self.age}")

c1=child("Sachin","kapoor",23)
c1.displaydetails()
```

```
My name is Sachin and age is kapoor
Age is 23
```

Important Hack:

In python there is a process called name mangling through which you can access the private data member outside the class as well.
In name mangling trailing underscore is textually replaced with _classname__identifier where classname is the name of the current class.

Let's access the private variable from outside the class.

In [241…
```
class Parent:
    def __init__(self,firstname,lastname):
        self.__firstname=firstname
        self.__lastname=lastname

    def showdetails(self):
        print(f"My name is {self.__firstname} {self.__lastname}")

class Child(Parent):
    def __init__(self,firstname,lastname,age):
        super().__init__(firstname,lastname)
        self.age=age
    def show(self):
        super().showdetails()
        print(f"and my age is {self.age}")

c1=Child("Sachin","Kapoor",23)
c1.show()
```

```
My name is Sachin Kapoor
and my age is 23
```

In [242…
```
print(p1._Person__age)
```

```
23
```

Polymorphism

As per the heading poly means "Many" and morph means "Forms".
The word polymorphism means having many forms. In programming, polymorphism means same function name (but different signatures) being uses for different types.

In python there are four type of polymorphism:

- Duck typing
- Method overriding
- Method overloading
- Operator overloading

Method overriding:

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

The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed. In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.

Example

In [243…
```
# Defining parent class
class Parent():

    # Constructor
    def __init__(self):
        self.value = "Inside Parent"

    # Parent's show method
    def show(self):
        print(self.value)

# Defining child class
class Child(Parent):

    # Constructor
    def __init__(self):
        self.value = "Inside Child"
```

```
    # Child's show method
    def show(self):
        print(self.value)


# Driver's code
obj1 = Parent()
obj2 = Child()

obj1.show()
obj2.show()
```

```
Inside Parent
Inside Child
```

Method overriding with multiple and multilevel inheritance

1. Multiple Inheritance: When a class is derived from more than one base class it is called multiple Inheritance.

Example: Let's consider an example where we want to override a method of one parent class only. Below is the implementation.

Example

In [244…

```
# Python program to demonstrate
# overriding in multiple inheritance


# Defining parent class 1
class Parent1:

    # Parent's show method
    def show(self):
        print("Inside Parent1")

# Defining Parent class 2
class Parent2:

    # Parent's show method
    def display(self):
        print("Inside Parent2")


# Defining child class
class Child(Parent1, Parent2):

    # Child's show method
    def show(self):
        print("Inside Child")


# Driver's code
obj = Child()

obj.show()
obj.display()
```

```
Inside Child
Inside Parent2
```

2. Multilevel Inheritance: When we have a child and grandchild relationship.

Example: Let's consider an example where we want to override only one method of one of its parent classes. Below is the implementation.

Example

In [245…

```
# Python program to demonstrate
# overriding in multilevel inheritance


# Python program to demonstrate
# overriding in multilevel inheritance


class Parent():

    # Parent's show method
    def display(self):
        print("Inside Parent")


# Inherited or Sub class (Note Parent in bracket)
class Child(Parent):

    # Child's show method
    def show(self):
        print("Inside Child")

# Inherited or Sub class (Note Child in bracket)
class GrandChild(Child):

    # Child's show method
    def show(self):
        print("Inside GrandChild")

# Driver code
g = GrandChild()
g.show()
g.display()
```

```
Inside GrandChild
Inside Parent
```

Calling the Parent's method within the overridden method Parent class methods can also be called within the overridden methods. This can generally be achieved by two ways.

1. Using Classname: Parent's class methods can be called by using the Parent classname.method inside the overridden method

In [246…

```
class Parent():

    def show(self):
        print("Inside Parent")

class Child(Parent):

    def show(self):

        # Calling the parent's class
        # method
        Parent.show(self)
        print("Inside Child")

# Driver's code
obj = Child()
obj.show()
```

```
Inside Parent
Inside Child
```

2. Using Super(): Python super() function provides us the facility to refer to the parent class explicitly. It is basically useful where we have to call superclass functions. It returns the proxy object that allows us to refer parent class by 'super'.

In [247…
```
class Parent():

    def show(self):
        print("Inside Parent")

class Child(Parent):

    def show(self):

        # Calling the parent's class
        # method
        super().show()
        print("Inside Child")

# Driver's code
obj = Child()
obj.show()
```

```
Inside Parent
Inside Child
```

Method Overloading

In python there is no such concept of method overloading by default. But there is a way to achieve this by using dispatch decorator

Example

In [248…
```
from multipledispatch import dispatch

#passing one parameter
@dispatch(int,int)
def product(first,second):
    result = first*second
    print(result);

#passing two parameters
@dispatch(int,int,int)
def product(first,second,third):
    result  = first * second * third
    print(result);

#you can also pass data type of any value as per requirement
@dispatch(float,float,float)
def product(first,second,third):
    result  = first * second * third
    print(result);


#calling product method with 2 arguments
product(2,3,2) #this will give output of 12
product(2.2,3.4,2.3) # this will give output of 17.985999999999997
```

```
12
17.204
```

Operator Overloading:

Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because '+' operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called Operator Overloading.

How to overload the operators in Python? Consider that we have two objects which are a physical representation of a class (user-defined data type) and we have to add two objects with binary '+' operator it throws an error, because compiler don't know how to add two objects. So we define a method for an operator and that process is called operator overloading. We can overload all existing operators but we can't create a new operator. To perform operator overloading, Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator. For example, when we use + operator, the magic method **add** is automatically invoked in which the operation for + operator is defined.

Overloading binary + operator in Python : When we use an operator on user defined data types then automatically a special function or magic function associated with that operator is invoked. Changing the behavior of operator is as simple as changing the behavior of method or function. You define methods in your class and operators work according to that behavior defined in methods. When we use + operator, the magic method **add** is automatically invoked in which the operation for + operator is defined. There by changing this magic method's code, we can give extra meaning to the + operator.

Example:

In [249…

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

     # adding two objects
    def __add__(self, other):
        return self.salary + other.salary


Ob1 = Employee("Sachin Kapoor", 15000)
Ob2 = Employee("Ranjeet Kumar", 25000)
Ob3 = Ob1 + Ob2
print(Ob3)
```

```
40000
```

Abstract classes:

An abstract class can be considered as a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class. A class which contains one or more abstract methods is called an abstract class. An abstract method is a method that has a declaration but does not have an

implementation. While we are designing large functional units we use an abstract class. When we want to provide a common interface for different implementations of a component, we use an abstract class.

Why use Abstract Base Classes : By defining an abstract base class, you can define a common Application Program Interface(API) for a set of subclasses. This capability is especially useful in situations where a third-party is going to provide implementations, such as with plugins, but can also help you when working in a large team or with a large code-base where keeping all classes in your mind is difficult or not possible.

How Abstract Base classes work : By default, Python does not provide abstract classes. Python comes with a module that provides the base for defining Abstract Base classes(ABC) and that module name is ABC. ABC works by decorating methods of the base class as abstract and then registering concrete classes as implementations of the abstract base. A method becomes abstract when decorated with the keyword @abstractmethod.

Example:

In [250…

```
from abc import ABC,abstractmethod

class Automobile(ABC):
    def __init__(self):
        print("Automobile created")

    def start(self):
        pass

    def stop(self):
        pass

    def drive(self):
        pass


A1=Automobile()
```

```
Automobile created
```

Code Explanation: From above code you can see you can create instance of abstract class.

In [251…

```
from abc import ABC,abstractmethod

class Automobile(ABC):
    def __init__(self):
        print("Automobile created")

    @abstractmethod
    def start(self):
        pass

    @abstractmethod
    def stop(self):
        pass

    @abstractmethod
    def drive(self):
        pass
```

```
A1=Automobile()
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-251-63342e10edb0> in <module>
     18
     19
---> 20 A1=Automobile()

TypeError: Can't instantiate abstract class Automobile with abstract meth
ods drive, start, stop
```

Code Explanation: From above code you have. If you can abstract method in abstract class then you can not instance of abstract class.

In [252…

```
from abc import ABC,abstractmethod

class Automobile(ABC):
    def __init__(self):
        print("Automobile created")

    @abstractmethod
    def start(self):
        pass

    @abstractmethod
    def stop(self):
        pass

    @abstractmethod
    def drive(self):
        pass


class Car(Automobile):
    def __init__(self):
        print("Car creaed")

C1=Car()
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-252-fa5b1d2a243a> in <module>
     22         print("Car creaed")
     23
---> 24 C1=Car()

TypeError: Can't instantiate abstract class Car with abstract methods dri
ve, start, stop
```

Code Explanation: From above code you can see. If you can inherit from abstract class you must have to implement all the abstract method in your child class. You must have to implement start,stop and drive method in car class.

In [253…

```
from abc import ABC,abstractmethod

class Automobile(ABC):
    def __init__(self):
        print("Automobile created")
```

```
    @abstractmethod
    def start(self):
        pass

    @abstractmethod
    def stop(self):
        pass

    @abstractmethod
    def drive(self):
        pass


class Car(Automobile):
    def __init__(self):
        print("Car created")

    def start(self):
        pass

    def stop(self):
        pass

    def drive(self):
        pass


C1=Car()
```

```
Car created
```

Code Explanation: Now you can see you have to implement all the abstract method start,stop,drive in car class.

In [254…

```
from abc import ABC,abstractmethod

class Automobile(ABC):
    def __init__(self):
        print("Automobile created")

    @abstractmethod
    def start(self):
        pass

    @abstractmethod
    def stop(self):
        pass

    @abstractmethod
    def drive(self):
        pass


class Car(Automobile):
    def __init__(self):
        print("Car created")

    def start(self):
        pass

    def stop(self):
        pass
```

```
    def drive(self):
        pass


C1=Car()
```

```
Car created
```

Code Explanation: When you have init method implementation in car class while instancing it will call init method of car class.

In [255…

```
from abc import ABC,abstractmethod

class Automobile(ABC):
    def __init__(self):
        print("Automobile created")

    @abstractmethod
    def start(self):
        pass

    @abstractmethod
    def stop(self):
        pass

    @abstractmethod
    def drive(self):
        pass


class Car(Automobile):
    def __init__(self):
        pass

    def start(self):
        pass

    def stop(self):
        pass

    def drive(self):
        pass


C1=Car()
```

Code Explanation: When you do not have init method implementation in car class while instancing it will call init method of abstract class.

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
class Vehicle:
    def __init__(self,color):
        self.color = color
class Car(Vehicle):
    def __init__(self,color,numGears):
        self.numGears = numGears
c= Car("black",5)
print(c.color)
```

Options

black

None

Error

None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class Vehicle:
    def __init__(self,color):
        self.color = color
class Car(Vehicle):
    def __init__(self,color,numGears):
        self.numGears = numGears
c= Car("black",5)
print(c.color)
```

Options

black

None

Error ✔

None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
class Vehicle:
    def __init__(self,color):
        self.color = color
class Car(Vehicle):
    def __init__(self,color,numGears):
        super().__init__(color)
        self.numGears = numGears
c= Car("black",5)
print(c.color)
```

Options

None

Error

black

None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class Vehicle:
    def __init__(self,color):
        self.color = color
class Car(Vehicle):
    def __init__(self,color,numGears):
        super().__init__(color)
        self.numGears = numGears
c= Car("black",5)
print(c.color)
```

Options

None

Error

black ✔

None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
class Vehicle:
    def __init__(self,color):
        self.__color = color
class Car(Vehicle):
    def __init__(self,color,numGears):
        super().__init__(color)
        self.numGears = numGears
    def printCar(self):
        print(c.__color,end=" )
        print(c.numGears)
c = Car("black",5)
c.printCar()
```

Options

black 5

Error

5 black

None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class Vehicle:
    def __init__(self,color):
        self.__color = color
class Car(Vehicle):
    def __init__(self,color,numGears):
        super().__init__(color)
        self.numGears = numGears
    def printCar(self):
        print(c.__color,end=" )
        print(c.numGears)
c = Car("black",5)
c.printCar()
```

Options

- ○ black 5
- ● Error ✔
- ○ 5 black
- ○ None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
class Vehicle:
    def __init__(self,color):
        self.color = color
    def print(self):
        print(c.color,end="")
class Car(Vehicle):
    def __init__(self,color,numGears):
        super().__init__(color)
        self.numGears = numGears
    def print(self):
        print(c.color,end="" )
        print(c.numGears)
c = Car("black",5)
c.print()
```

Options

black 5

Error

black

None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class Vehicle:
   def __init__(self,color):
      self.color = color
   def print(self):
      print(c.color,end="")
class Car(Vehicle):
   def __init__(self,color,numGears):
      super().__init__(color)
      self.numGears = numGears
   def print(self):
     print(c.color,end="" )
     print(c.numGears)
c = Car("black",5)
c.print()
```

Options

- black 5 ✔
- Error
- black
- None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
class Vehicle:
   def __init__(self,color):
      self.color = color
   def print(self):
      print(c.color,end="")
class Car(Vehicle):
   def __init__(self,color,numGears):
      super().__init__(color)
      self.numGears = numGears
   def print(self):
      self.print()
      print(c.numGears)
c = Car("black",5)
c.print()
```

Options

- black 5
- Recursion Error
- black
- None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class Vehicle:
   def __init__(self,color):
      self.color = color
   def print(self):
      print(c.color,end="")
class Car(Vehicle):
   def __init__(self,color,numGears):
      super().__init__(color)
      self.numGears = numGears
   def print(self):
      self.print()
      print(c.numGears)
c = Car("black",5)
c.print()
```

Options

- black 5
- Recursion Error ✔
- black
- None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
class A:

    def __init__(self):
        print("init of A called")
class B:
    def __init__(self):
        print("init of B called")

class C(B,A):
    def __init__(self):
        super().__init__()

c = C()
```

Options

- init of A called
- init of B called
- Nothing will be printed
- None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
class A:

    def __init__(self):
        print("init of A called")
class B:
    def __init__(self):
        print("init of B called")

class C(B,A):
    def __init__(self):
        super().__init__()

c = C()
```

Options

- ○ init of A called
- ● init of B called ✔
- ○ Nothing will be printed
- ○ None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
from abc import ABC,abstractmethod

class A(ABC):

    @abstractmethod
    def fun1(self):
        pass

    @abstractmethod
    def fun2(self):
        pass

o = A()
o.fun1()
```

Options

- ○ Nothing will be printed.
- ○ Error
- ○ None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

```
from abc import ABC,abstractmethod

class A(ABC):

    @abstractmethod
    def fun1(self):
        pass

    @abstractmethod
    def fun2(self):
        pass

o = A()
o.fun1()
```

Options

- Nothing will be printed.
- Error ✔
- None of the above

Correct Answer

Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
from abc import ABC,abstractmethod

class A(ABC):

    @abstractmethod
    def fun1(self):
        pass

    @abstractmethod
    def fun2(self):
        pass

class B(A):

    def fun1(self):
        print("function 1 called")
o = B()
o.fun1()
```

Options

- function 1 called
- Nothing will be printed.
- Error
- None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

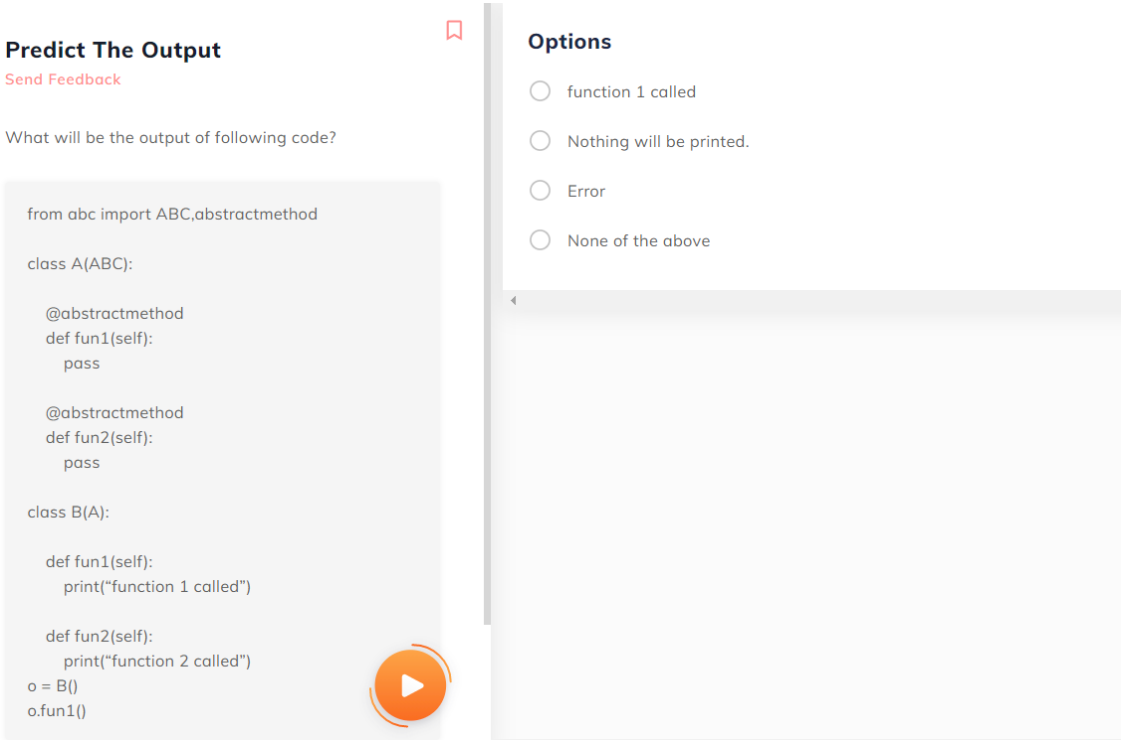
```
from abc import ABC,abstractmethod

class A(ABC):

    @abstractmethod
    def fun1(self):
        pass

    @abstractmethod
    def fun2(self):
        pass

class B(A):

    def fun1(self):
        print("function 1 called")
o = B()
o.fun1()
```

Options

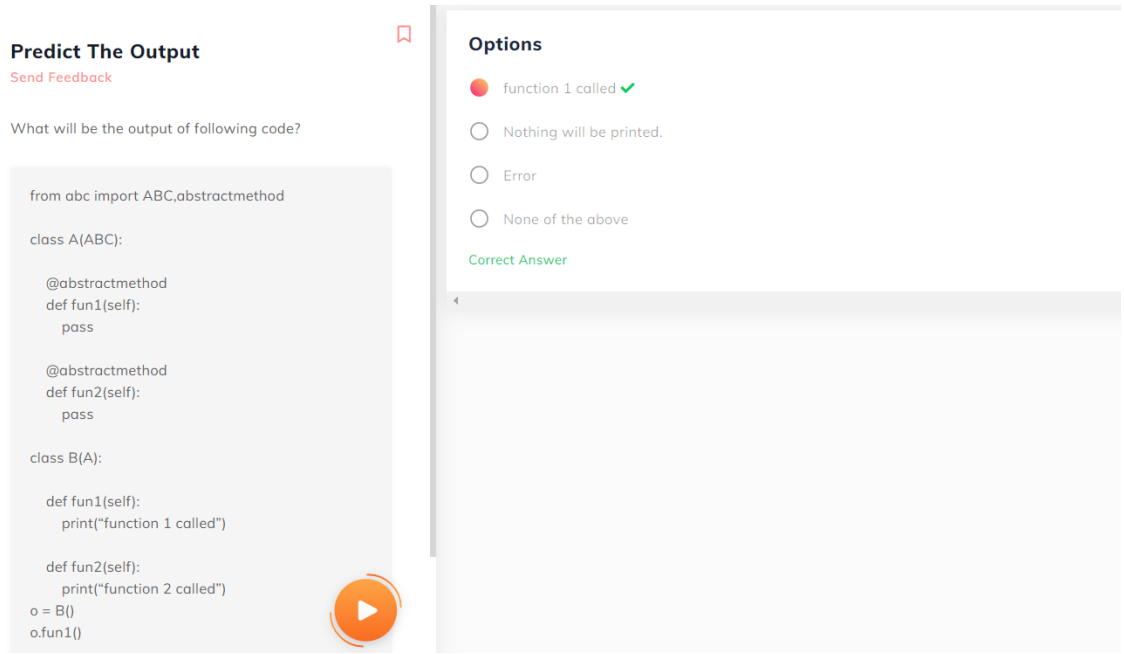
- function 1 called
- Nothing will be printed.
- Error ✔
- None of the above

Correct Answer

Quiz



Solution



Quiz

Predict The Output

Send Feedback

What will be the output of following code?

```
from abc import ABC,abstractmethod

class A(ABC):

    @abstractmethod
    def fun1(self):
        print("function of class A called")

    @abstractmethod
    def fun2(self):
        pass

class B(A):
    def fun1(self):
        print("function 1 called")
    def fun2(self):
        print("function 2 called")
o = B()
o.fun1()
```

Options

- function 1 called
- function of class A called
- Error
- None of the above

Solution

Predict The Output

Send Feedback

What will be the output of following code?

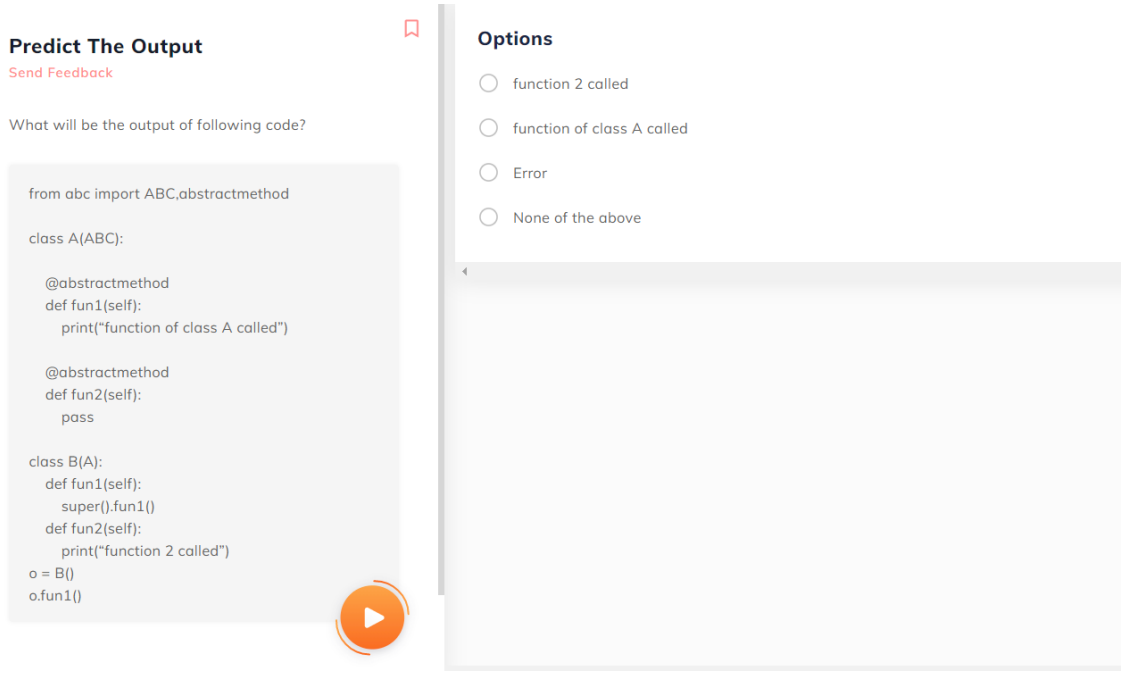
```
from abc import ABC,abstractmethod

class A(ABC):

    @abstractmethod
    def fun1(self):
        print("function of class A called")

    @abstractmethod
    def fun2(self):
        pass

class B(A):
    def fun1(self):
        print("function 1 called")
    def fun2(self):
        print("function 2 called")
o = B()
o.fun1()
```

Options

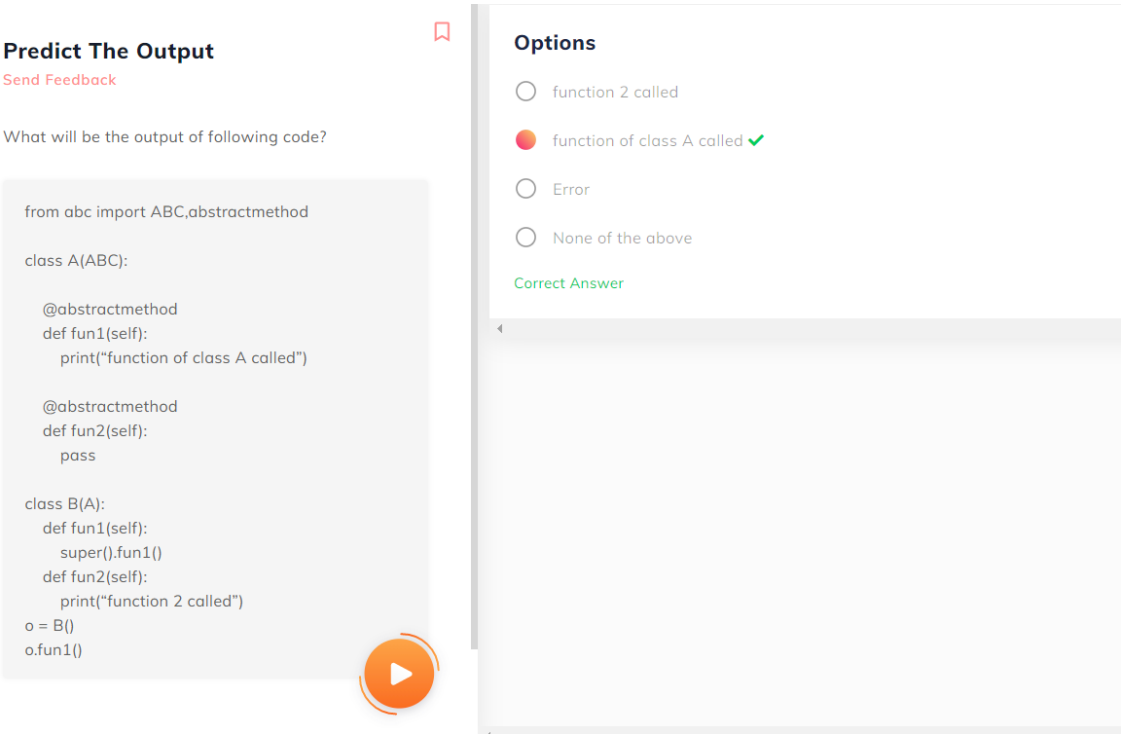
- function 1 called ✔
- function of class A called
- Error
- None of the above

Correct Answer

Quiz



Solution



==========

Error

The first thing you need to know about programming is how to print **"Hello, world!"**. The second one is that this might be a challenging task, as even such a tiny script can contain various errors. Here you are:

In [256…

```
print("Hello, world!"
```

```
  File "<ipython-input-256-ec501c8f224c>", line 1
    print("Hello, world!"
```

```
SyntaxError: unexpected EOF while parsing
```

Traceback is a stack trace that appears when your code causes an error and it reports detailed information on that particular error, indicating the definite files in which the error occurred. Nonetheless, the lines that are the most informative for us right now are the last two. They point out the mistake in your code.

Common errors for beginners

Some of the most common syntax errors are:

- wrong spelling of keywords and function names, e.g. While instead of while, pint instead of print
- The wrong number of parentheses in function calls, e.g. print "just one round bracket");
- Indents are also the fertile soil for errors, therefore, use spaces and tabs carefully;
- quote Don't forget to wrap a string in quotes of the same type: triple quotes for multi-line strings, double or single quotes for ordinary strings

Exception handling:

Exception is an event, which occurs during the execution of a program that disrupts the normal flow of program instructions. When a python scripts raises an exception. It must either handle the exception immediately otherwise it terminates and quits.

- 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.

Example

In [257…
```
try:
    print(var)
except:
    print("An error exception occurred")
```

```
23
```

Code Explanation: Since We are trying to print the value of x variable but we did not define the x variable so it will raise error the try block raise an error, the except block will be executed. Without the try block, the program will crash and raise an error.

Many exception: You can define as many as exception blocks as you want.
e.g., if you want to execute a special block of code for a special kind of error.

Example

In [307…
```
try:
    print(varnew)
except NameError:
    print("Variable x is not defined")
```

```
except:
    print("Something else went wrong")
```

```
Variable x is not defined
```

else: You can use the else keyword to define a block of code to be executed if no errors were raised.

Example

In [308…

```
try:
    print("Hello")
except NameError:
    print("Something went wrong")
except:
    print("Nothing went wrong")
```

```
Hello
```

finally: The finally block, if specified will be executed regardless if the try block raises an error or not.

Example

In [309…

```
try:
    print(varnew)
except NameError:
    print("Something went wrong")
except:
    print("The try except is finished")
finally:
    print("You are in finally block")
```

```
Something went wrong
You are in finally block
```

Raise an exception: As a python developer you can choose to throw an exception if a condition occurs.
To throw an exception use the raise keyword.

Example

In [261…

```
x=-1
if x<0:
    raise exception("sorry, no numbers below zero is accepted")
```

```
---------------------------------------------------------------------------
--
NameError                                 Traceback (most recent call las
t)
<ipython-input-261-447a0c321371> in <module>
      1 x=-1
      2 if x<0:
----> 3     raise exception("sorry, no numbers below zero is accepted")

NameError: name 'exception' is not defined
```

Code Explain: The raise keyword is used to raise an exception you can define what kind of errors to raise, and the test to print the user.

User-defined exceptions:

However, some programs may need user-defined exceptions for their special needs. Let's

consider the following example. We need to add two integers but we do not want to work with negative integers. Python will process the addition correctly, so we can create a custom exception to raise it if any negative numbers appear. So, it is necessary to know how to work with the user-created exceptions along with the built-in ones.

1. Raising user-defined exceptions

If we want our program to stop working when something goes wrong, we can use the raise keyword for the Exception class when a condition is met. You may come across either your or built-in exceptions like the ZeroDivisionError in the example below. Note that you can specify your feedback in brackets to explain the error. It will be shown when the exception occurs.

Example

In [262…
```
def example_exceptions_1(x, y):
    if y == 0:
        raise ZeroDivisionError("The denominator is 0! Try again, please
    elif y < 0:
        raise Exception("The denominator is negative!")
    else:
        print(x / y)
```

In [263…
```
example_exceptions_1(10, 0)
```

```
---------------------------------------------------------------------------
--
ZeroDivisionError                         Traceback (most recent call las
t)
<ipython-input-263-9f97bb626e07> in <module>
----> 1 example_exceptions_1(10, 0)

<ipython-input-262-b58314c7fbe1> in example_exceptions_1(x, y)
      1 def example_exceptions_1(x, y):
      2     if y == 0:
----> 3         raise ZeroDivisionError("The denominator is 0! Try again,
please!")
      4     elif y < 0:
      5         raise Exception("The denominator is negative!")

ZeroDivisionError: The denominator is 0! Try again, please!
```

In [264…
```
example_exceptions_1(10, -2)
```

```
---------------------------------------------------------------------------
--
Exception                                 Traceback (most recent call las
t)
<ipython-input-264-59c01de1acf0> in <module>
----> 1 example_exceptions_1(10, -2)

<ipython-input-262-b58314c7fbe1> in example_exceptions_1(x, y)
      3         raise ZeroDivisionError("The denominator is 0! Try again,
please!")
      4     elif y < 0:
----> 5         raise Exception("The denominator is negative!")
      6     else:
      7         print(x / y)

Exception: The denominator is negative!
```

In [265…
```
example_exceptions_1(10, 5)
```

```
2.0
```

Code Explain: If there is a zero, the program will stop working and will display the built-in exception with the message you specified; note that if we had not raised this exception ourselves, it would have been raised by Python but with a regular message. If y is a negative integer, we get the user-defined exception and the message. If the integer is positive, it prints the results.

2. Creating a user-defined exception class

we are creating a new class of exceptions named NegativeResultError derived from the built-in Exception class.

We print the message informing that the procedure of creating the class is finished inside.

NOTE: Note that it is good to end the name of the exception class with such word as Error or Exception

In [266…
```
class NegativeResultError(Exception):
    print("Hooray! My first exception is working!")

def example_exceptions_2(a, b):
    try:
        c = a / b
        if c < 0:
            raise NegativeResultError
        else:
            print(c)
    except NegativeResultError:
        print("There is a negative result!")
```

```
Hooray! My first exception is working!
```

In [267…
```
example_exceptions_2(2, 5)
```

```
0.4
```

In [268…
```
example_exceptions_2(2, -5)
```

```
There is a negative result!
```

Code Explanation: In the example_exceptions_2(a, b) function below we use the try-except block. If the result of the division is positive, we just print the result. If it is negative, we raise an exception and go to the corresponding part of the code with except to print the message.

==========

File Handling

As far whatever we have learnt. We were using the RAM(volatile memory) to store the variable but there is one problem with volatile memory. Whenever we shut down our machine all the variable get cleaned from machine or machine itself erase all the Variable/Data from RAM.

But let's us assume if we want our data to be present lifetime till i delete from my machine. Data should be present there.

So we have file handling concept for that. Whatever data we are going to store it goes to Hard disk(non-volatile memory) it will remain there until user delete it from there end.

To solve the query to store data in Hard disk File Handling come in the picture to solve

this.

File handling is an important part of any web application.
Python has several functions for creating, reading, updating, and deleting files.

How to open a file: We use open function to open a file.
The open() function takes two parameters; filename, and mode.

Syntax:

In [ ]:
```
variable_name=open("File_name","file_mode")
```

Mode of file: There are four different methods (modes) for opening a file:

- "r" - Read - Default value. Opens a file for reading, error if the file does not exist.
- "a" - Append - Opens a file for appending, creates the file if it does not exist.
- "w" - Write - Opens a file for writing, creates the file if it does not exist.
- "x" - Create - Creates the specified file, returns an error if the file exists.

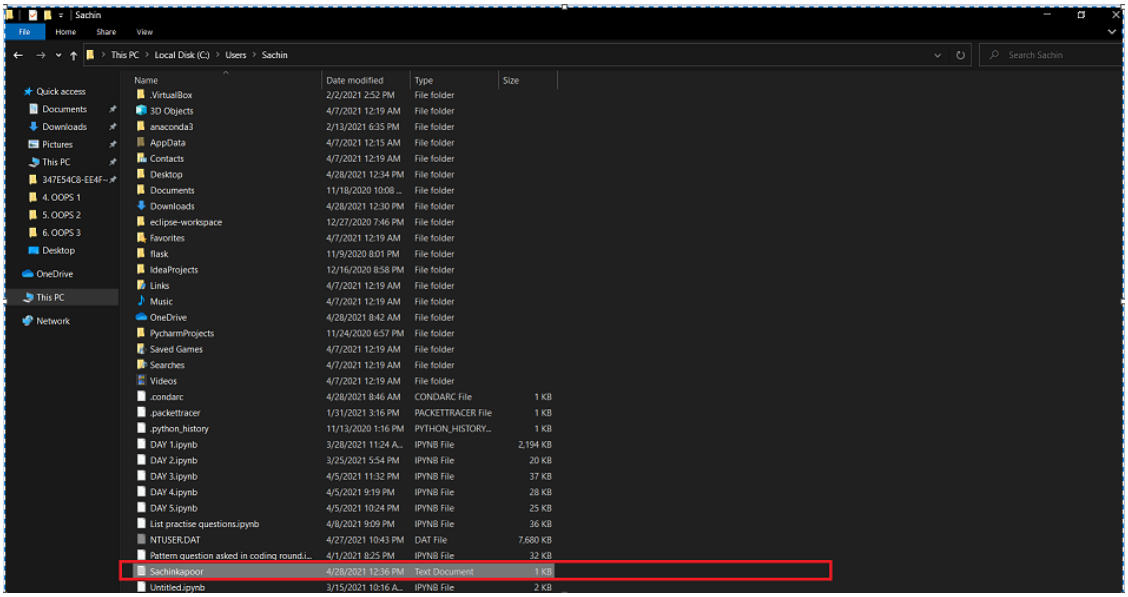
Type of file: In addition you can specify if the file should be handled as binary or text mode

- "t" - Text - Default value. Text mode.
- "b" - Binary - Binary mode (e.g. images).

To open a file for reading it is enough to specify the name of the file:

Example:

In [270…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt")
```



I have open this "Sachinkapoor.txt" file which is present in my local machine.

NOTE: When you open any file its by default mode is "r" and file type is "t".
The code above is the same as:

f = open(r"C:\Users\Sachin\Sachinkapoor.txt", "rt")

Open and Read from a file:

To open the file, use the built-in open() function.

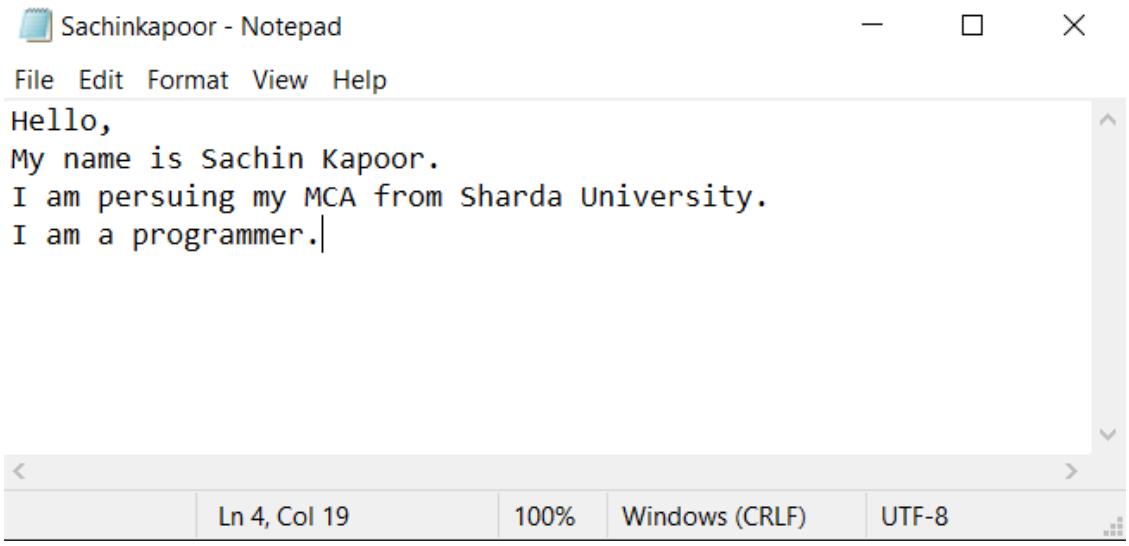
read(): We use read() method for reading the content of the file.

Example:

In [271…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","rt")    #open "Sachinkapoor
print(f.read())
```

```
Hello,
My name is Sachin Kapoor.
I am persuing my MCA from Sharda University.
I am a programmer.
```

Content of my file:



NOTE: If we open a file in read mode and file doesn't exist it will return error.

In [274…
```
f = open(r"C:\Users\Sachin\Sachinkapoor1.txt","rt")    #open "Sachinkapoo
print(f.read())
```

```
---------------------------------------------------------------------------
--
FileNotFoundError                         Traceback (most recent call las
t)
<ipython-input-274-cdca372fdc58> in <module>
----> 1 f = open(r"C:\Users\Sachin\Sachinkapoor1.txt","rt")    #open "Sach
inkapoor" file in text file type and read mode
      2 print(f.read())

FileNotFoundError: [Errno 2] No such file or directory: 'C:\\Users\\Sachi
n\\Sachinkapoor1.txt'
```

Code Explanation: We don't have any file name "sachin1.txt" in my machine that's why python interpreter not able to find. So, it's return an error.

Open file in binary mode: You just need to mentioned the file type as "b" in open function.

Example:

In [275…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","rb")  #open "Sachinkapoor"

print(f.read())
```

```
b'Hello,\r\nMy name is Sachin Kapoor.\r\nI am persuing my MCA from Sharda
University.\r\nI am a programmer.'
```

Read Only Parts of the File: By default the read() method returns the whole text, but you can also specify how many characters you want to return:

Example:

In [276…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","rt")  #open "Sachinkapoor"

print(f.read(20))
```

```
Hello,
My name is Sa
```

In [277…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","rt")  #open "Sachinkapoor"

print(f.read(20))
print(f.read(20))
```

```
Hello,
My name is Sa
chin Kapoor.
I am pe
```

Read Lines: You can return one line by using the readline() method:

Example:

In [278…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","rt")  #open "Sachinkapoor"

print(f.readline())
```

```
Hello,
```

In [279…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","rt")  #open "Sachinkapoor"

print(f.readline())
print(f.readline())
```

```
Hello,

My name is Sachin Kapoor.
```

Print line by line: By looping through the lines of the file, you can read the whole file, line by line.

Example:

In [280…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","rt")

for line in f:
    print(line)
```

```
Hello,

My name is Sachin Kapoor.

I am persuing my MCA from Sharda University.

I am a programmer.
```

print lines in list: You can return the list of lines by using readlines() function.

Example:

In [281…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","rt")
print(f.readlines())
```

```
['Hello,\n', 'My name is Sachin Kapoor.\n', 'I am persuing my MCA from Sharda University.\n', 'I am a programmer.']
```

Close Files: It is a good practice to always close the file when you are done with it.

Example:

In [282…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","rt")
print(f.read())
f.close()
```

```
Hello,
My name is Sachin Kapoor.
I am persuing my MCA from Sharda University.
I am a programmer.
```

Write in a file:

write() : We use write() method to write in an existing file.

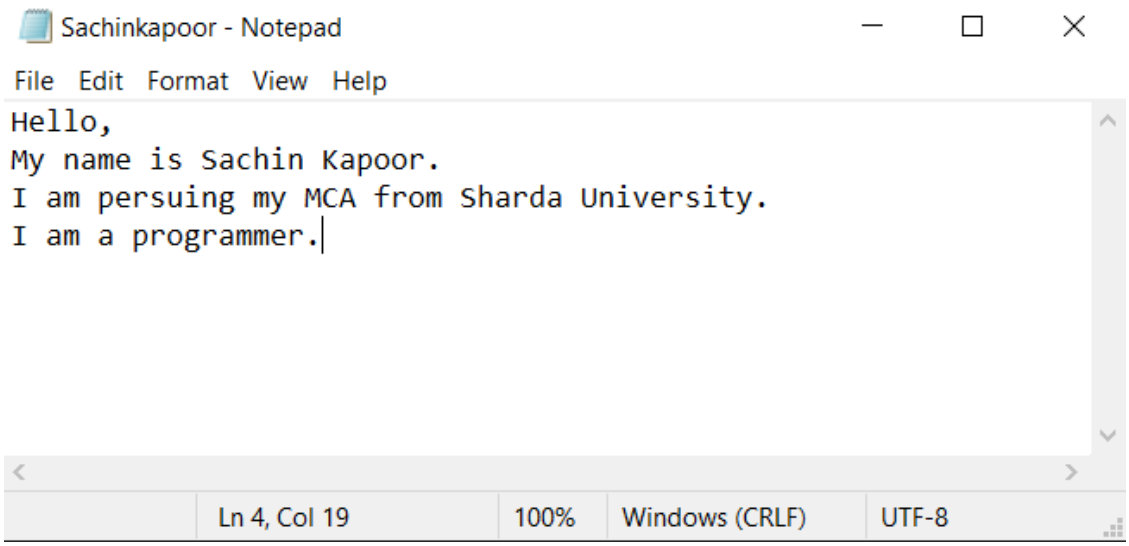
To write to an existing file, you must add a parameter to the open() function:

- "a" - Append - will append to the end of the file.
- "w" - Write - will overwrite any existing content.

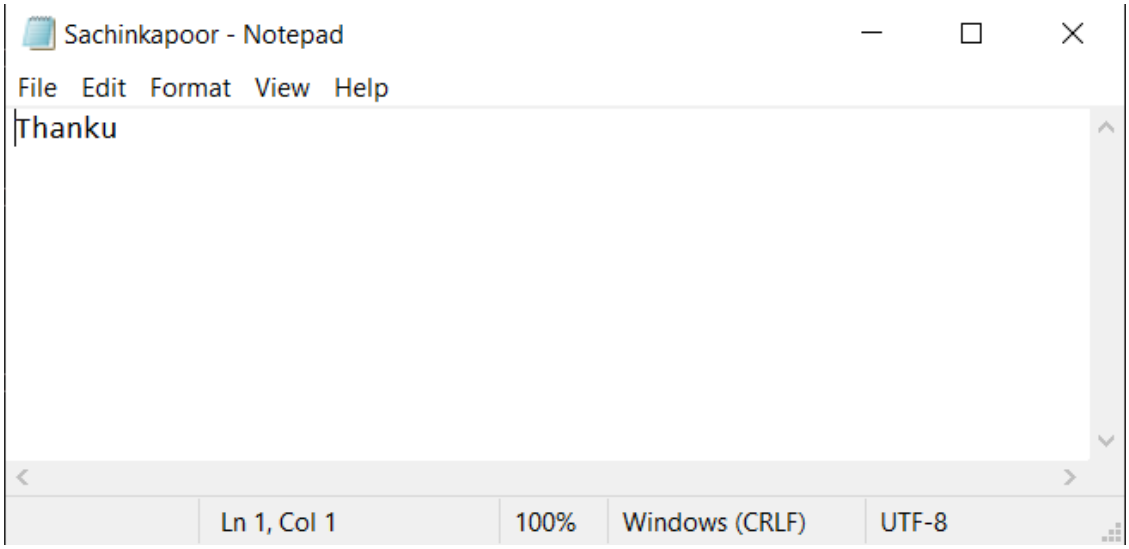
Example:

In [283…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","w")
f.write("Thanku")
f.close()
```

Before running this code the content of my file was:



After running this code the content of my file is:

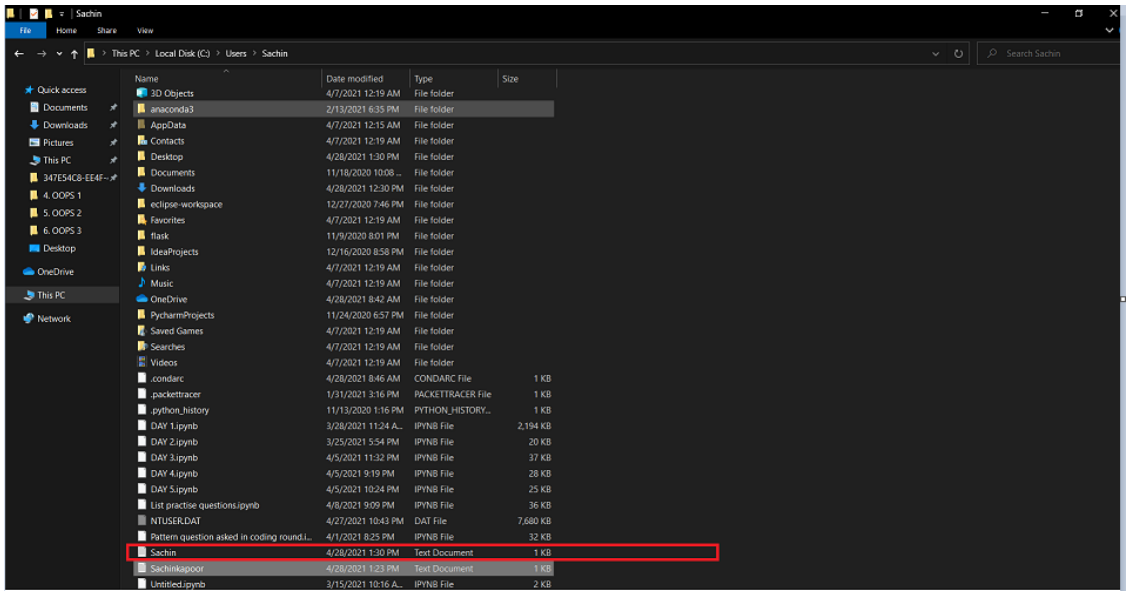


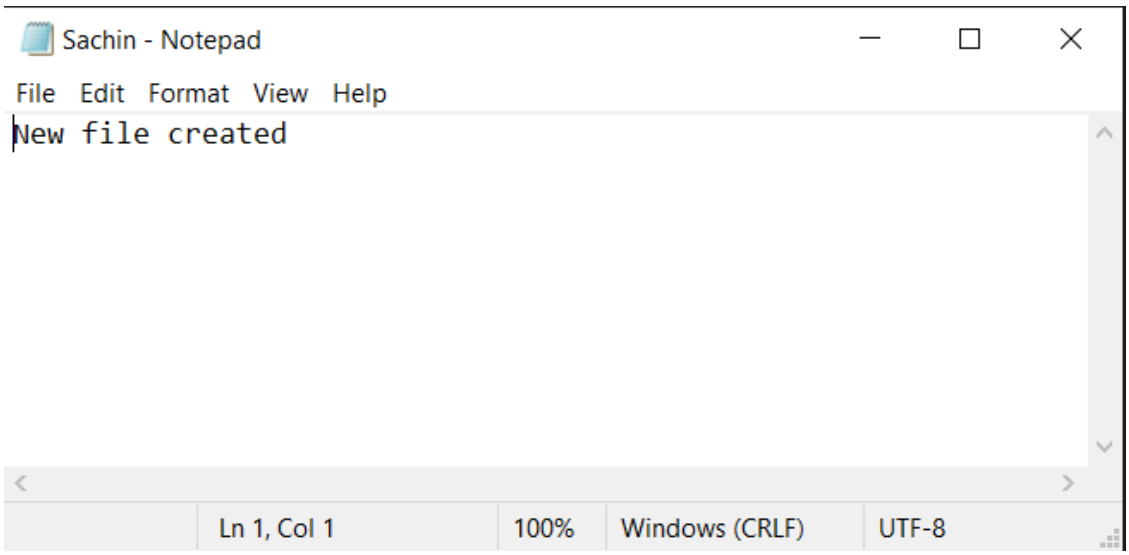
NOTE: When you write anything in file using write() method it will erase all the data first and then append to the list whatever you want.

NOTE: If we open a file in write mode and file doesn't exist it will create file.

In [284…

```
f = open(r"C:\Users\Sachin\Sachin.txt","w")
f.write("New file created")
f.close()
```





See it creates new file an append the data whatever we have written in it.

Example:

In [285…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","a")
f.write("Hey programmers")
f.close()

f = open(r"C:\Users\Sachin\Sachinkapoor.txt","r")
print(f.read())
f.close()
```

```
ThankuHey programmers
```

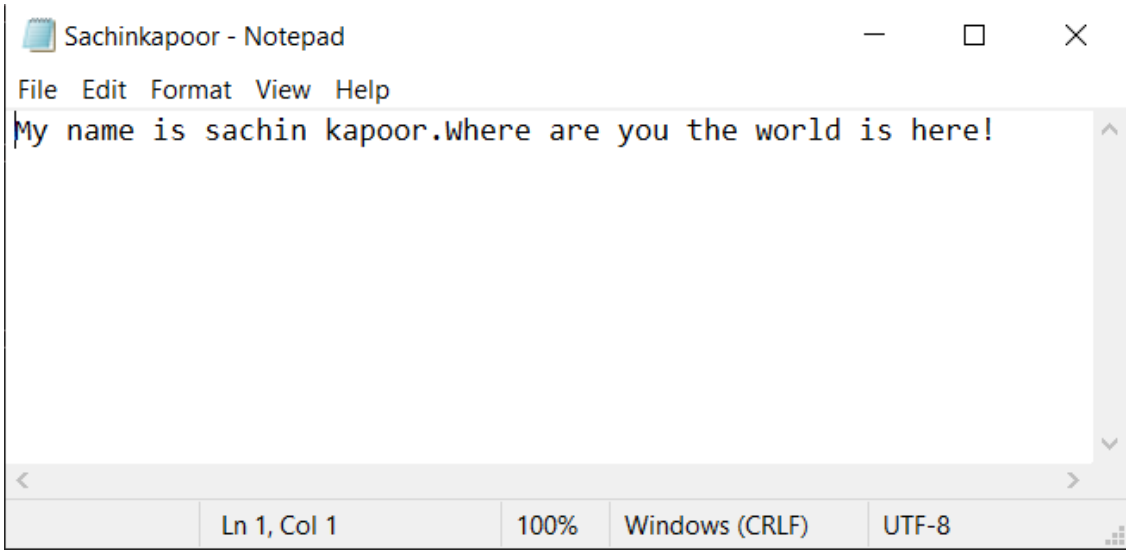
NOTE: If we use append file mode whatever we will write it will append to the end of the file.

Handle read and write both

Example:

In [286…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","r+")
print(f.read())
f.write("Where are you the world is here!")
print(f.read())
f.close()
```

```
ThankuHey programmers
```



tell(): The tell() method returns the current file position in a file stream.

NOTE: You can change the current file position with the seek() method.

Example:

In [287…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","r")

print(f.tell())
print(f.readline())
print(f.tell())
```

```
0
ThankuHey programmersWhere are you the world is here!
53
```

seek(): The seek() method sets the current file position in a file stream.
The seek() method also returns the new postion.

Example:

In [288…
```
f = open(r"C:\Users\Sachin\Sachinkapoor.txt","r")

print(f.read(25))
f.seek(0)
print(f.read())
```

```
ThankuHey programmersWher
ThankuHey programmersWhere are you the world is here!
```

Mini Project Health Management System

Problem statement you have three clients Sachin kapoor, Abhishek Jaiswal, Ranjeet Kumar. You have to create two seperate file for each client in one file is for exercise and another one is for diet track. You can track the each client data.

In [291…
```
print("Choose the client")
print("1. Abhishek Jaiswal")
print("2. Ranjeet kumar")
print("3. Sachin kapoor")
client=int(input("Enter the client number: "))
print()
print("Choose the mode of the file")
print("1. To retreive the data")
print("2. To add the data")
mode=int(input("Enter the mode: "))
print()
print("Choose which file you want")
print("1. Diet File")
print("2. Exercise File")
file=int(input("Enter which file you want to choose: "))
print()

if(client==1):
    abhishekdiet = open(r"C:\Users\Sachin\Abhishekdiet.txt","at")
    abhishekexercise = open(r"C:\Users\Sachin\Abhishekexercise.txt","at"
    if(mode==1 and file==1):
        print("your diet list is here")
        abhishekdiet = open(r"C:\Users\Sachin\Abhishekdiet.txt","rt")
        print(abhishekdiet.read())
    elif(mode==1 and file==2):
        print("Your exercise list is here: \n")
        abhishekexercise = open(r"C:\Users\Sachin\Abhishekexercise.txt",
        print(abhishekexercise.read())
    elif(mode==2 and file==1):
        diet=input("Write into diet file: \n")
        abhishekdiet.write(diet)
        print("Written successfully in diet file")
    elif(mode==2 and file==2):
        exercise=input("Write into exercise file: \n")
        abhishekexercise.write(exercise)
        print("Written successfully in exercise file")
    abhishekdiet.close()
    abhishekexercise.close()


elif(client==2):
```

```
    ranjeetdiet = open(r"C:\Users\Sachin\Ranjeetdiet.txt","at")
    ranjeetexercise = open(r"C:\Users\Sachin\Ranjeetexercise.txt","at")
    if(mode==1 and file==1):
        print("your diet list is here")
        ranjeetdiet = open(r"C:\Users\Sachin\Ranjeetdiet.txt","rt")
        print(ranjeetdiet.read())
    elif(mode==1 and file==2):
        print("Your exercise list is here: \n")
        ranjeetexercise = open(r"C:\Users\Sachin\Ranjeetexercise.txt","r
        print(ranjeetexercise.read())
    elif(mode==2 and file==1):
        diet=input("Write into diet file: \n")
        ranjeetdiet.write(diet)
        print("Written successfully in diet file")
    elif(mode==2 and file==2):
        exercise=input("Write into exercise file: \n")
        ranjeetexercise.write(exercise)
        print("Written successfully in exercise file")
    ranjeetdiet.close()
    ranjeetexercise.close()


elif(client==3):
    sachindiet = open(r"C:\Users\Sachin\Sachindiet.txt","at")
    sachinexercise = open(r"C:\Users\Sachin\Sachinexercise.txt","at")
    if(mode==1 and file==1):
        print("your diet list is here")
        sachindiet = open(r"C:\Users\Sachin\Sachindiet.txt","rt")
        print(sachindiet.read())
    elif(mode==1 and file==2):
        print("Your exercise list is here: \n")
        sachinexercise = open(r"C:\Users\Sachin\Sachinexercise.txt","rt"
        print(sachinexercise.read())
    elif(mode==2 and file==1):
        diet=input("Write into diet file: ")
        sachindiet.write(diet)
        print("Written successfully in diet file")
    elif(mode==2 and file==2):
        exercise=input("Write into exercise file")
        sachinexercise.write(exercise)
        print("Written successfully in exercise file")
    sachindiet.close()
    sachinexercise.close()
```

```
Choose the client
1. Abhishek Jaiswal
2. Ranjeet kumar
3. Sachin kapoor
Enter the client number: 3

Choose the mode of the file
1. To retreive the data
2. To add the data
Enter the mode: 1

Choose which file you want
1. Diet File
2. Exercise File
Enter which file you want to choose: 2

Your exercise list is here:

dumbell presschest fly
```

==========

Some Important Function You Need To Know

Filter Function:

The filter() function takes two arguments, function object and an iterable. This function filters out all elements in iteable, for which that function returns True.Note that, filter function can take only one iterable as input.

Syntax:

In [ ]:
```
filter(function,iterables)
```

Example:

In [292…
```
li = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

lst = list(filter(lambda x: (x % 2 == 0), li))

print(lst)
```

```
[2, 4, 6, 8, 10]
```

Map Function:

The map() functions takes a function object and iterables like sequence, collection or an iterator object.You can send as many iterables as you like, just make sure the function has one parameter for each iterable. It executes the function object for each element in the sequence and returns a map object which we can convert it to list using list().

Syntax:

In [ ]:
```
filter(function,iterables)
```

Example:

In [293…
```
li = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

lst = list(map(lambda x: x+2, li))

print(lst)
```

```
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Reduce Function

The reduce() function takes a function and an iterable and returns a single value.This performs a repetitive operation over the pairs of the iterable.This function is defined in functools module.

Syntax:

In [ ]:
```
reduce(function, iterable)
```

Example:

In [294…
```
from functools import reduce
li = [1, 2, 3, 4, 5]
```

```
sum = reduce((lambda x, y: x + y), li)
print(sum)
```

```
15
```

Zip Function

The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.

NOTE: If the passed iterators have different lengths, the iterator with the least items decides the length of the new iterator.

Syntax:

In [ ]:
```
zip(iterator1, iterator2, iterator3 ...)
```

Example:

In [295…
```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica", "Vicky")

x = zip(a, b)
print(list(x))
```

```
[('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica')]
```

In [296…
```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica", "Vicky")

x = zip(a, b)
print(tuple(x))
```

```
(('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica'))
```

In [297…
```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica", "Vicky")

x = zip(a, b)
print(set(x))
```

```
{('Charles', 'Christy'), ('Mike', 'Monica'), ('John', 'Jenny')}
```

In [298…
```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica", "Vicky")

x = zip(a, b)
print(dict(x))
```

```
{'John': 'Jenny', 'Charles': 'Christy', 'Mike': 'Monica'}
```

Searching and Sorting

Searching:

Searching means to find out whether a particular element is present in a given sequence or not.

There are commonly two types of searching techniques:

- Linear search
- Binary search

Linear Search

Linear search is the simplest searching algorithm that searches for an element in a list in sequential order.

Linear Search Algorithm

We have been given a list and a targetValue. We aim to check whether the given targetValue is present in the given list or not. If the element is present we are required to print the index of the element in the list and if the element is not present we print -1.

The following are the steps in the algorithm:

1. Traverse the given list using a loop.
2. In every iteration, compare the targetValue with the value of the element in the list in the current iteration.
   - If the values match, print the current index of the list.
   - If the values do not match, move on to the next list element.
3. If no match is found, print -1.

Pseudo-Code

In [ ]:
```
for each element in the array:#For loop to traverse the list
    if element == targetValue #Compare the targetValue to the element
        print(indexOf(item))
```

Code:

In [300…
```
li= [int(x) for x in input().split(" ")] #Taking list as user input
targetValue= int(input()) #User input for targetValue

found = False #Boolean value to check if we found the targetValue
for i in li:
    if (i==targetValue): #If we found the targetValue
        print(li.index(i)) #Print the index
        found = True #Set found as True as we found the targetValue
        break #Since we found the targetValue, we break out of loop
if found is False:#If we did not find the targetValue
    print(-1)
```

```
10 22 69 45 100 98
45
3
```

Binary Search

Binary Search is a searching algorithm for finding an element's position in a sorted array. In a nutshell, this search algorithm takes advantage of a collection of elements of being already sorted by ignoring half of the elements after just one comparison.

Prerequisite: Binary search has one pre-requisite; unlike the linear search where elements could be any order, the array in **binary** search must be sorted,

The algorithm works as follows:

1. Let the element we are searching for, in the given array/list is X.

2. Compare X with the middle element in the array.
3. If X matches with the middle element, we return the middle index.
4. If X is greater than the middle element, then X can only lie in the right (greater) half subarray after the middle element, then we apply the algorithm again for the right half.
5. If X is smaller than the middle element, then X must lie in the left (lower) half, this is because the array is sorted. So we apply the algorithm for the left half.

Example Run

● Let us consider the array to be:

● Let x = 4 be the element to be searched.
● Set two pointers low and high at the first and the last element respectively.
● Find the middle element mid of the array ie. arr[(low+high)/2] = 6.

● If x == mid, then return mid. Else, compare the element to be searched with m.
● If x > mid, compare x with the middle element of the elements on the right side of mid. This is done by setting low to low = mid + 1.
● Else, compare x with the middle element of the elements on the left side of mid. This is done by setting high to high = mid - 1.

● Repeat these steps until low meets high. We found 4:

Code:

In [303…

```
#Function to implement Binary Search Algorithm
def binarySearch(array, x, low, high):

    #Repeat until the pointers low and high meet each other
    while low <= high:
        mid = low + (high - low) #Middle Index
        if array[mid] == x: #Element Found
            return mid
        elif array[mid] < x: #x is on the right side
            low = mid + 1
        else: #x is on the left side
            high = mid - 1
    return -1 #Element is not found

array = [3, 4, 5, 6, 7, 8, 9]
x = 6

result = binarySearch(array, x, 0, len(array)-1)

if result != -1: #If element is found
    print("Element is present at index ",(result))
else: #If element is not found
    print("Not found")
```

```
Element is present at index  3
```

Advantages of Binary search:

- This searching technique is faster and easier to implement.
- Requires no extra space.
- Reduces the time complexity of the program to a greater extent. (The term time complexity might be new to you, you will get to understand this when you will be studying algorithmic analysis. For now, just consider it as the time taken by a particular algorithm in its execution, and time complexity is determined by the number of operations that are performed by that algorithm i.e. time complexity is directly proportional to the number of operations in the program).

Sorting

Sorting is a permutation of a list of elements of such that the elements are either in increasing (ascending) order or decreasing (descending) order.

There are many different sorting techniques. The major difference is the amount of space and time they consume while being performed in the program.

For now, we will be discussing the following sorting techniques:

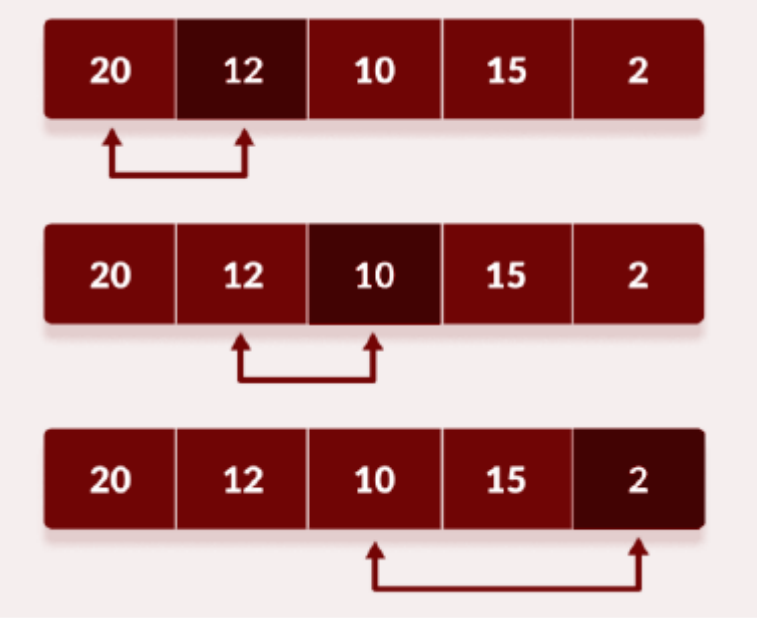
- Selection sort
- Bubble sort
- Insertion sort

Selection Sort

Selection sort is an algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list. The detailed algorithm is given below.

- Consider the given unsorted array to be:

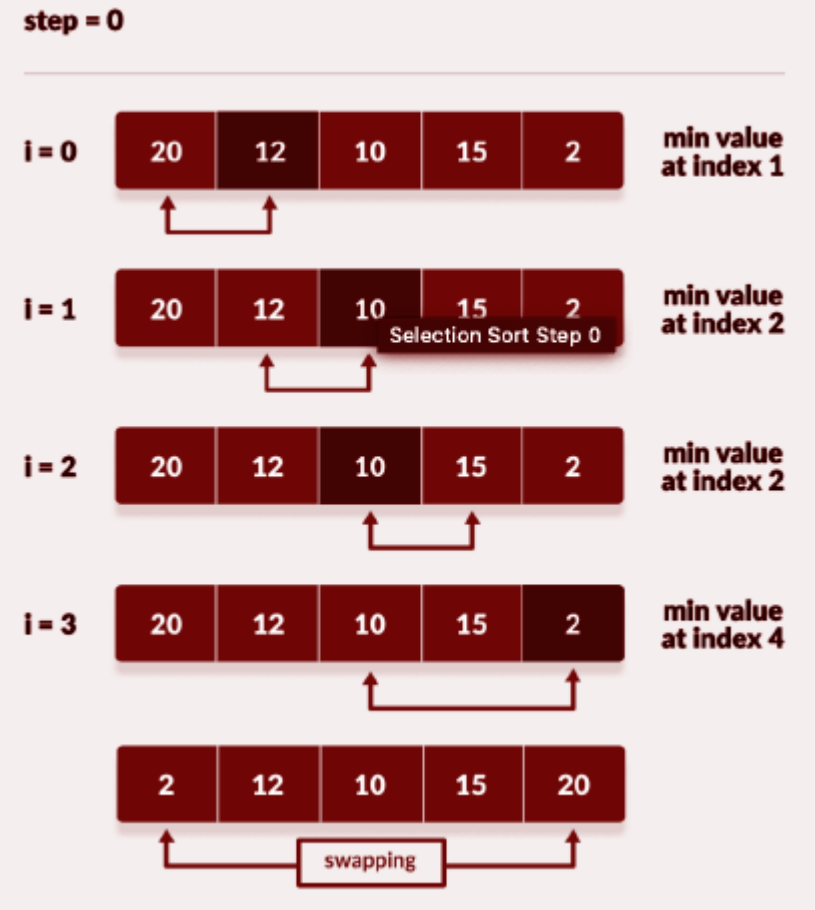
- Set the first element as **minimum**.
- Compare **minimum** with the second element. If the second element is smaller than **minimum**, assign the second element as **minimum**.
- Compare **minimum** with the third element. Again, if the third element is smaller, then assign **minimum** to the third element otherwise do nothing. The process goes on until the last element.



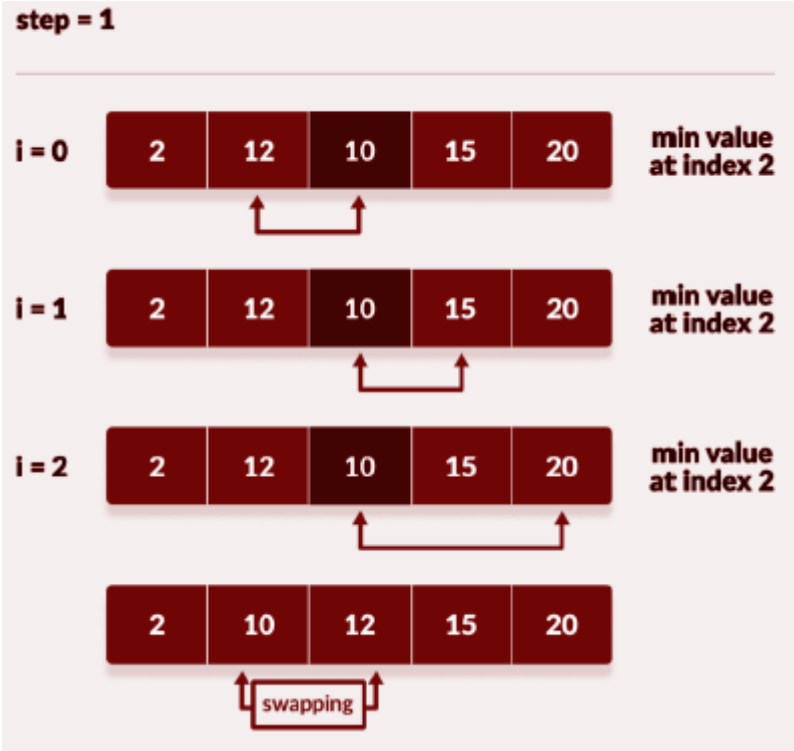
- After each iteration, minimum is placed in the front of the unsorted list.

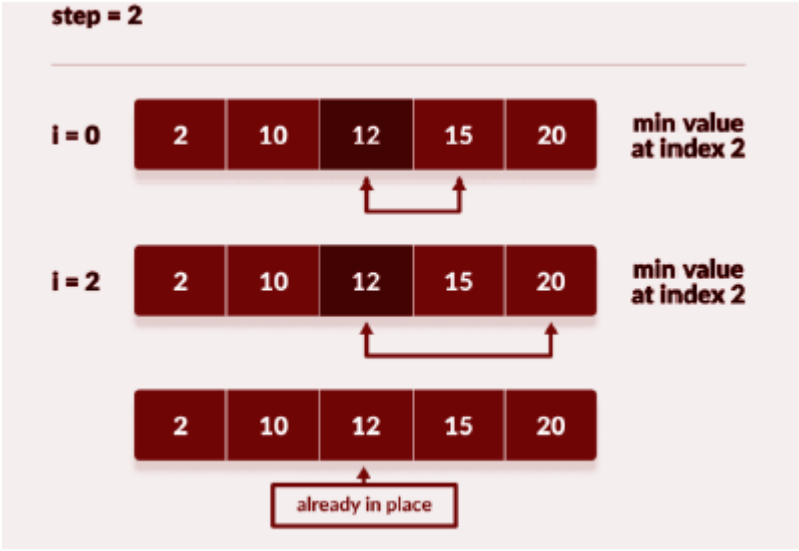
- For each iteration, indexing starts from the first unsorted element. These steps are repeated until all the elements are placed at their correct positions.

First Iteration:

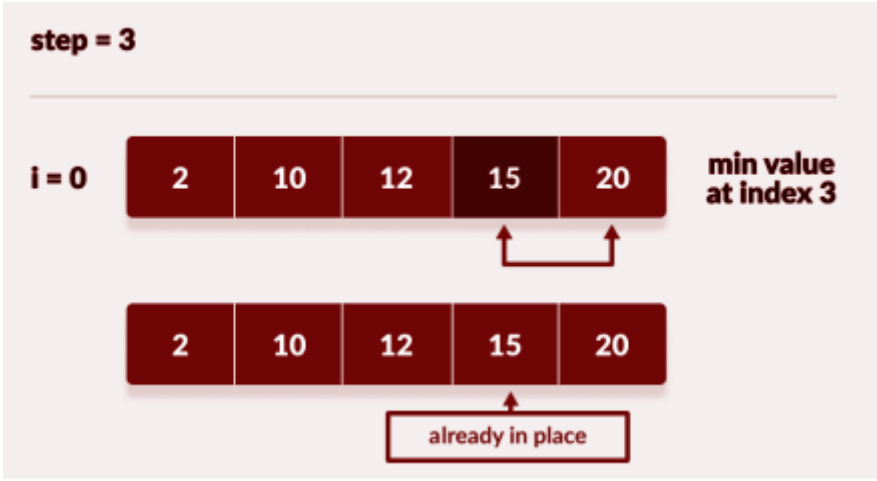


Second Iteration:





Third Iteration:



Fourth Iteration:

Code:

In [304…

```
# Selection sort in Python
def selectionSort(arr, size):

    for step in range(size):
        minimum = step

        for i in range(step + 1, size):

            # to sort in descending order, change > to < in this line
            # select the minimum element in each loop
            if arr[i] < arr[minimum]:
                minimum = i
        # Swap
        (arr[step], arr[minimum]) = (arr[minimum], arr[step])

input = [20, 12, 10, 15, 2]
size = len(input)
selectionSort(input, size)
print('Sorted Array in Ascending Order: ')
print(input)
```

```
Sorted Array in Ascending Order:
[2, 10, 12, 15, 20]
```

Bubble Sort

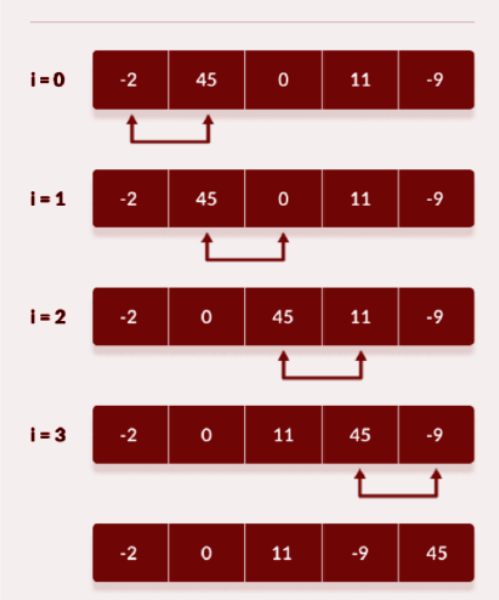
Bubble sort is an algorithm that compares the adjacent elements and swaps their

positions if they are not in the intended order. The order can be ascending or descending.

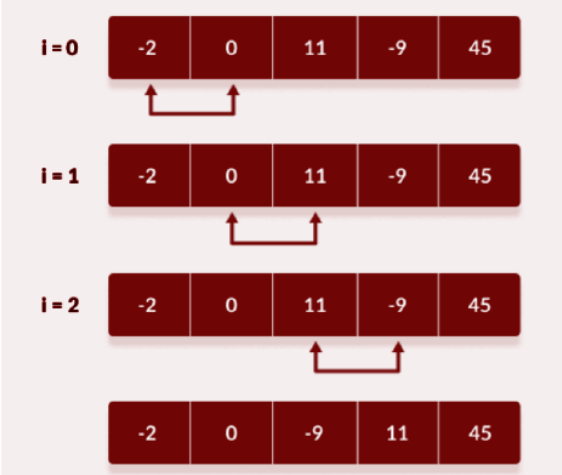
How does Bubble Sort work?

● Starting from the first index, compare the first and the second elements. If the first element is greater than the second element, they are swapped.
● Now, compare the second and third elements. Swap them if they are not in order.
● The above process goes on until the last element.
● The same process goes on for the remaining iterations. After each iteration, the largest element among the unsorted elements is placed at the end.
● In each iteration, the comparison takes place up to the last unsorted element.
● The array is sorted when all the unsorted elements are placed at their correct positions.

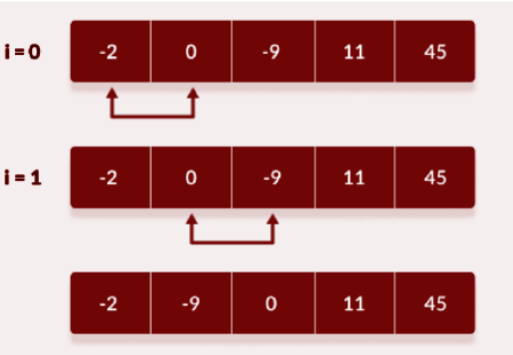
Let the array be [-2, 45, 0, 11, -9]. First



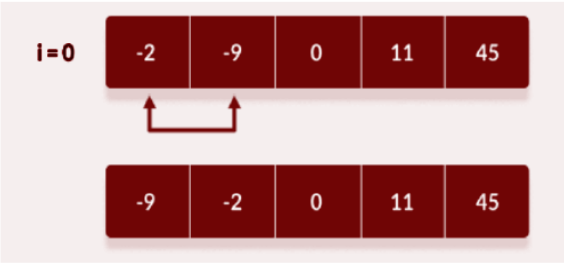
First Iteration



Second Iteration



Third Iteration



Fourth Iteration

Code:

In [305…

```
# Bubble sort in Python
def bubbleSort(arr):
    # run loops two times: one for walking through the array
    # and the other for comparison
    for i in range(len(arr)):
        for j in range(0, len(arr) - i - 1):
            # To sort in descending order, change > to < in this line.
```

```
            if arr[j] > arr[j + 1]:
                # swap if greater is at the rear position
                (arr[j], arr[j + 1]) = (arr[j + 1], arr[j])

input = [-2, 45, 0, 11, -9]
bubbleSort(input)
print('Sorted Array in Ascending Order:')
print(input)
```

```
Sorted Array in Ascending Order:
[-9, -2, 0, 11, 45]
```

Insertion Sort

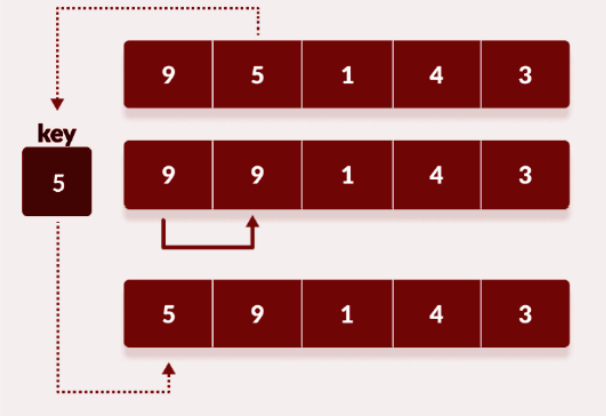
● Insertion sort works similarly as we sort cards in our hand in a card game.
● We assume that the first card is already sorted.
● Then, we select an unsorted card.
● If the unsorted card is greater than the card in hand, it is placed on the right otherwise, to the left.
● In the same way, other unsorted cards are taken and put in the right place. A similar approach is used by insertion sort.
● Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.

ALGORITHM

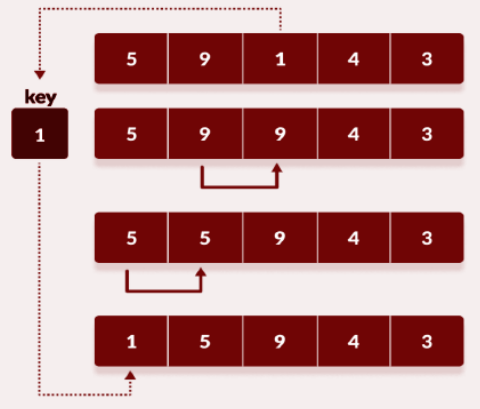
● Suppose we need to sort the following array.

● The first element in the array is assumed to be sorted. Take the second element and store it separately in key.
● Compare key with the first element. If the first element is greater than key, then key is placed in front of the first element.
● If the first element is greater than key, then key is placed in front of the first element.
● Now, the first two elements are sorted.
● Take the third element and compare it with the elements on the left of it. Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.
● Similarly, place every unsorted element at its correct position.

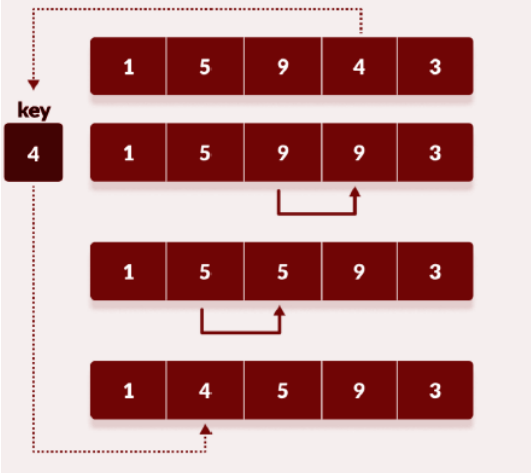
The various iterations are depicted below:



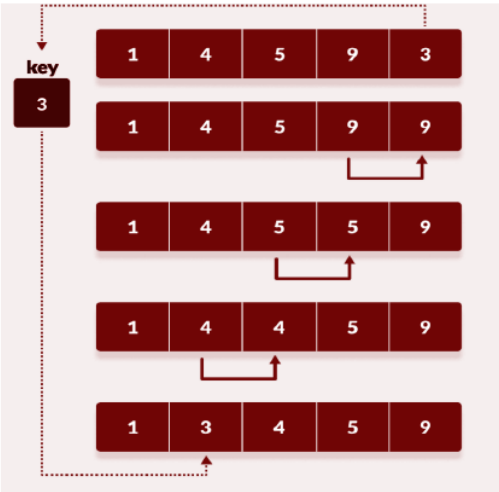
First Iteration



Second Iteration



Third Iteration



Fourth Iteration

Code:

In [306…

```
# Insertion sort in Python
def insertionSort(arr):

    for step in range(1, len(arr)):
        key = arr[step]
        j = step - 1

# Compare key with each element on the left of it until an element small
        # For descending order, change key<array[j] to key>array[j].
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j = j - 1

        # Place the key at after the element just smaller than it.
        arr[j + 1] = key

input = [9, 5, 1, 4, 3]
insertionSort(input)
print('Sorted Array in Ascending Order:')
print(input)
```

```
Sorted Array in Ascending Order:
[1, 3, 4, 5, 9]
```