

import my-module.py

message = my-module.info("Junaid")

print(message)

~~OP~~
~~Hello Junaid~~

\$ python main.py

O/P

Hello Junaid!

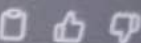
Description

- we created "my-module.py" and wrote our code
- we imported "my-module.py" to "main.py"
- then we wrote ~~out~~ ~~that~~ python statements directly
- then we used '\$ python main.py' to run our code



explain in detail about classes, self-variable, methods, constructor methods in python

In Python, a class is a blueprint for creating objects, which are instances of that class. Classes provide a way to define and organize data and the functions (methods) that operate on that data. Let's dive into the details of classes, self-variable, methods, and constructor methods in Python:



1. Classes:

- A class is defined using the `class` keyword, followed by the class name.
- Classes typically contain data attributes (variables) and methods (functions) that define the behavior of objects created from the class.
- Objects created from a class are instances of that class and have access to its attributes and methods.

python

Copy code

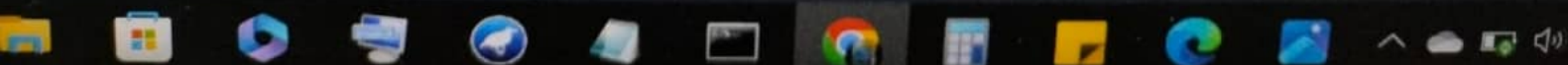
```
class MyClass:  
    # Data attributes  
    attribute1 = "Hello"  
    attribute2 = 42
```

Regenerate

Send a message



Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. [ChatGPT August 3 Version](#)



Example

```
# m-script.py
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
total = num1 + num2
print("Sum of {num1} and {num2} is {total}")

# Importing my-script.py
# main.py
$ python my-script.py
```

O/P

Enter first number: 5

Enter second number: 2

Sum of 5 and 2 is 7

(iii) module mode

- > In module mode we open a new text editor or IDE named "my-module.py"
- > Then we write our code in "my-module.py"
- > Then we import "my-module.py" to main.py
- > Then we enter our python statements directly.

Example

```
#my-module.py
def info(name):
    return f"Hello, {name}!"

if __name__ == '__main__':
```



```
print(module1.module_variable) # Accessing module_variable from module1
print(module2.module_variable) # Accessing module_variable from module2

module1.module_function() # Calling module_function from module1
module2.module_function() # Calling module_function from module2
```

In this example:

- `module1` and `module2` have their own separate namespaces, each containing a `module_variable` and a `module_function`.
- In `main.py`, we import both `module1` and `module2`, allowing us to access and use the names defined in each module.

Namespace conflicts can occur if two imported modules have names with the same identifier. In such cases, you can use aliases to avoid conflicts:

python

Copy code

```
import module1 as m1
import module2 as m2

print(m1.module_variable) # Using an alias to access module1's variable
print(m2.module_variable) # Using an alias to access module2's variable
```

This way, you can differentiate between names in different namespaces.

Regenerate

Default (GPT-3.5)

between the two namespaces, allowing you to access names defined in the imported module.

For example, consider the following Python modules:

module1.py:

python

Copy code

```
# This is module1.py
module_variable = 42

def module_function():
    print("This is a function in module1")
```

module2.py:

python

Copy code

```
# This is module2.py
module_variable = "Hello, World!"

def module_function():
    print("This is a function in module2")
```

Regenerate

Send a message

Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT August 3 Version

Relational database (such as PostgreSQL or MySQL)

ID	FIRST_NAME	LAST_NAME	PHONE
1	John	Connor	+16105551234
2	Matt	Makai	+12025555689
3	Sarah	Smith	+19735554512
...

Python objects

```
class Person:  
    first_name = "John"  
    last_name = "Connor"  
    phone_number = "+16105551234"
```

```
class Person:  
    first_name = "Matt"  
    last_name = "Makai"  
    phone_number = "+12025555689"
```

```
class Person:  
    first_name = "Sarah"  
    last_name = "Smith"  
    phone_number = "+19735554512"
```

ORMs provide a bridge between
**relational database tables, relationships
and fields** and **Python objects**

```
# This is main.py
import module1
import module2


print(module1.module_variable) # Accessing module_variable from module1
print(module2.module_variable) # Accessing module_variable from module2

module1.module_function() # Calling module_function from module1
module2.module_function() # Calling module_function from module2
```

In this example:

- `'module1'` and `'module2'` have their own separate namespaces, each containing a `'module_variable'` and a `'module_function'`.
- In `'main.py'`, we import both `'module1'` and `'module2'`, allowing us to access and use the names defined in each module.

Namespace conflicts can occur if two imported modules have names with the same identifier. In such cases, you can use aliases to avoid conflicts:

 Regenerate

Python Unit-1 Sem
Q How do you run python script in different modes. Explain with a python program.

Ans: we can run python scripts in various modes like

- (i) Interactive mode
- (ii) script mode
- (iii) module mode
- (iv) command line arguments mode

(i) Interactive mode

python
→ In interactive mode you can enter statements directly

Example

```
print("my name is junaid")
```

OP

```
my name is junaid
```

Description

Here we have directly entered our python statement in python shell which gave us our output directly

(ii) Script mode

In script mode we open a text editor or IDE to create a new file named "my-script.py"

then we write our code in "my-script.py"

→ Then we import "my-script.py"

→ Then we enter our python statements directly

8. Write Program to print the number of lines, words and characters present in the given file.

The basic idea is to traverse each line in a file and count the number of words and characters.

[data.txt](#)

[main.py](#)

```
1 number_of_words = 0
2 number_of_lines = 0
3 number_of_characters = 0
4
5 with open("data.txt", 'r') as file:
6     for l in file:
7         number_of_words += len(l.split())
8         number_of_lines += 1
9         number_of_characters = len(l)
10
11 print("No of words: ", number_of_words)
12 print("No of lines: ", number_of_lines)
13 print("No of characters: ", number_of_characters)
```

Output

```
No of words: 784
No of lines: 14
No of characters: 998
```

In the above code snippet:

- **Lines 1–4:** We declare and initialize variables with value `0` to store the total count of words, characters, and lines.
- **Line 5:** We open the file in reading mode `r`.
- **Line 6:** We loop through each line in the file using the `for` loop.
- **Line 7:** We get the list of words present in a line using the `split()` method. We calculate the length/number of words passing the result of the `split()` method to the `len()` function and add it to the `number_of_words` variable.

- **Line 8:** We add value `1` to `number_of_lines` variable as we progress through each line in a file.
- **Line 9:** We get the count of characters using the `len()` function and add the result to the `number_of_characters` variable.

Once we traverse each line in a file, we get the count of lines, words, and characters.

7. How will you create a Package & import it? Explain it with an example program

We organize a large number of files in different folders and subfolders based on some criteria, so that we can find and manage them easily. In the same way, a package in

Python takes the concept of the modular approach to next logical level. As you know, a module can contain multiple objects, such as classes, functions, etc. A package can contain one or more relevant modules. Physically, a package is actually a folder containing one or more module files.

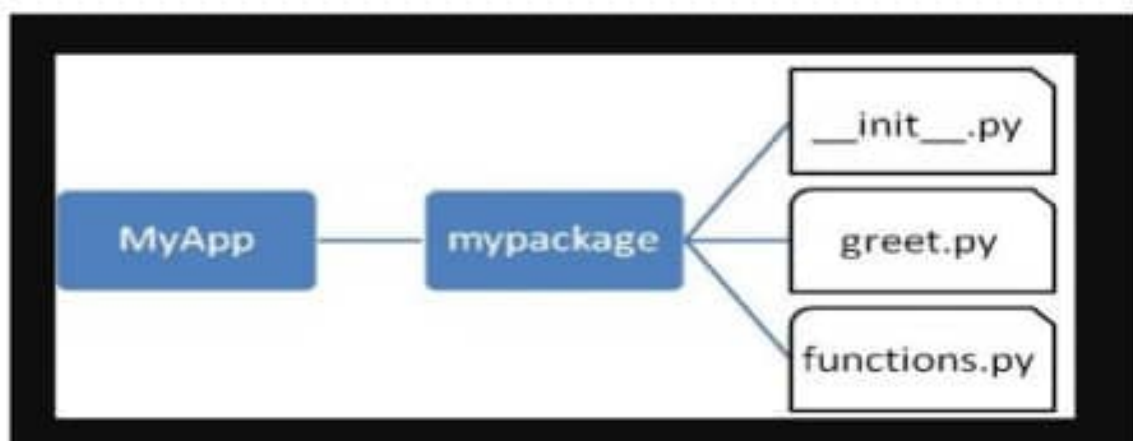
Let's create a package named mypackage, using the following steps:

- Create a new folder named D:\MyApp.
- Inside MyApp, create a subfolder with the name 'mypackage'.
- Create an empty `__init__.py` file in the mypackage folder.
- Using a Python-aware editor like IDLE, create modules `greet.py` and `functions.py` with the following code:

```
greet.py  
  
def SayHello(name):  
    print("Hello ", name)
```

```
functions.py  
  
def sum(x,y):  
    return x+y  
  
def average(x,y):  
    return (x+y)/2  
  
def power(x,y):  
    return x**y
```

That's it. We have created our package called mypackage. The following is a folder structure:



Importing Module From a Package in Python

Packages help in ensuring the reusability of code. To access any module or file from a Python package, use an import statement in the Python source file where you want to access it. Import of modules from Python Packages is done using the dot operator (.). Importing modules and packages helps us make use of existing functions and code that can speed up our work.


In Python, a namespace is a container that holds a set of identifiers (such as variable names, function names, class names, etc.) and maps them to their corresponding objects.

Namespaces are used to avoid naming conflicts and to organize the code in a modular and structured way. Python has several types of namespaces, including module namespaces, function namespaces, and class namespaces.

Here, we'll focus on module namespaces:

1. **Module Namespace:**

- A module is a Python script or a file containing Python code, often organized around a specific functionality or purpose.
- Every module has its own namespace, which acts as a container for all the names (variables, functions, classes, etc.) defined within that module.
- Module namespaces are isolated from each other, meaning that names defined in one module won't interfere with names in another module unless explicitly imported.
- When you import a module into another module using `'import'`, you create a link between the two namespaces, allowing you to access names defined in the imported module.

 Regenerate

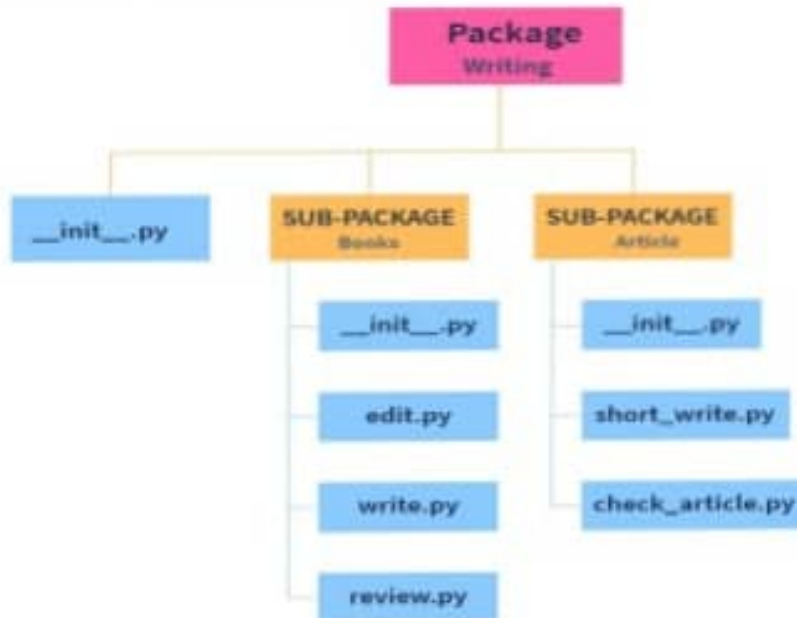


Syntax:

```
import module1[, module2,... moduleN]
```

Where import is a keyword used to import the module, module1 is the module to be imported. The other modules enclosed in brackets are optional and can be mentioned only when more than 1 module is to be imported.

Consider the writing package given below.



To use the edit module from the writing package in a new file test.py that you have created, you write the following code in the test.py;

```
import Writing.Book.edit
```

To access a function called plagiarism_check() of the edit module, you use the following code:

```
Writing.Book.edit.plagiarism_check()
```

The calling method seems lengthy and confusing, right? Another way to import a module would be to simply import the package prefix instead of the whole package and call the function required directly.

```
from Writing.Book import edit
```

```
plagiarism_check()
```

However, the above method may cause problems when 2 or more packages have similarly named functions, and the call may be ambiguous. Thus, this method is avoided in most cases.



-bfc108569f6d

ets | Video Notebook



wynk music



Messages-phone



Find My Device

behavior of objects created from the class.

- Objects created from a class are instances of that class and have access to its attributes and methods.

python



Copy c

```
class MyClass:
    # Data attributes
    attribute1 = "Hello"
    attribute2 = 42

    # Methods
    def my_method(self):
        return "This is a method"
```

2. Self-Variable:

- In Python, the first parameter of a method is usually named `'self'`. It represents the instance of the class and is automatically passed when calling methods on objects.
- It allows you to access and manipulate the object's attributes within methods.
- While `'self'` is the convention, you can name it differently, but it's a best practice to stick with `'self'` for clarity.



Regener

fc108569f6d



| Video Notebook



wynk music



Messages-phone



Find My Device

```
class MyClass:
    def set_attribute(self, value):
        self.attribute = value

    def get_attribute(self):
        return self.attribute
```

3. Methods:

- Methods are functions defined within a class.
- They operate on the object's data and can also perform actions related to the object.
- Methods often use the `'self'` parameter to access and modify object attributes.
- You can define instance methods (accessing object-specific data), class methods (operating on class-level data), and static methods (not dependent on instance or class data).

python

Copy code

```
class Calculator:
    def add(self, x, y):
        return x + y
```

Regenerate

08569f6d

| Video Notebook | wynk music | Messages-phone | Find My Device

```
class Calculator:
    def add(self, x, y):
        return x + y

    @staticmethod
    def multiply(a, b):
        return a * b

    @classmethod
    def square(cls, x):
        return x ** 2
```

4. Constructor Methods:

- Constructor methods are special methods used for initializing objects when they are created from a class.
- The constructor method in Python is named `__init__`, and it's automatically called when an object is created.
- It allows you to set up initial attributes or perform any other setup needed for the object.

python

Copy code

```
class Person:
```

Regenerate

end a message

python

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        return f"My name is {self.name} and I am {self.age} years old."

# Creating an object and initializing it using the constructor
person1 = Person("Alice", 30)
```

Copy code

Here's how you can use the class and objects:

python

```
# Creating an object of MyClass
obj = MyClass()

# Accessing attributes and calling methods using the object
obj.set_attribute("Hello, World!")
print(obj.get_attribute()) # Output: Hello, World!
```

Copy code

Regenerate

Send a message

Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT August 3 Version

