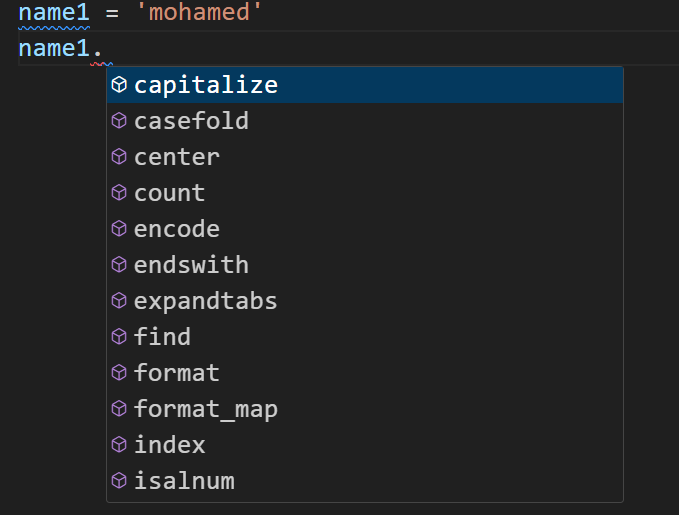
**A python interpreter :**

* is a program that converts the code written in python language by the user to the language which computer hardware or system can understand

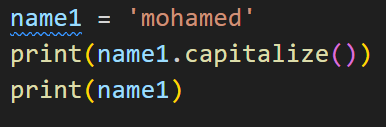
**Function vs Method:**

* Function is doing something General like … Len()
* Method is doing something related to specific thing like … methods manipulating strings

We can see methods related to string by writing (name of string then dot ).



These Methods doesn't affect our string , it does create another one and manipulate it

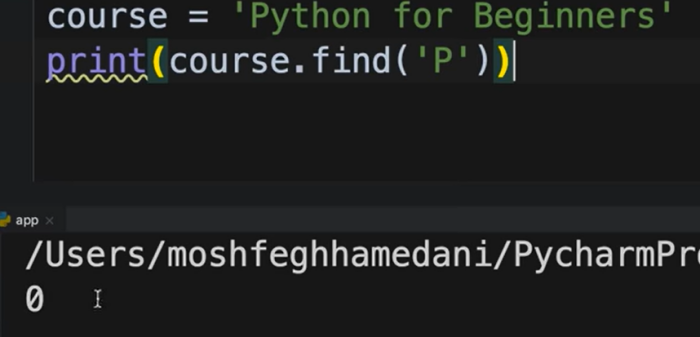




**Find()** :

- search for the index of the a character in a string and return the first occurrence

- General purpose Function in python

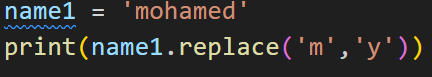


Output = 0

**Replace()** :

- replace a group of character with another one

- This method is Case sensitive , if we typed ‘M’ it wont find it





**In** :

- we use it to check the existence of a string

- Its Case sensitive

**Example :**

Will search for ‘mohamed’ and return boolen value



**input()** :

- the function takes a prompt (an optional string) as an argument, displays it to the user, and then waits for the user to enter some text.

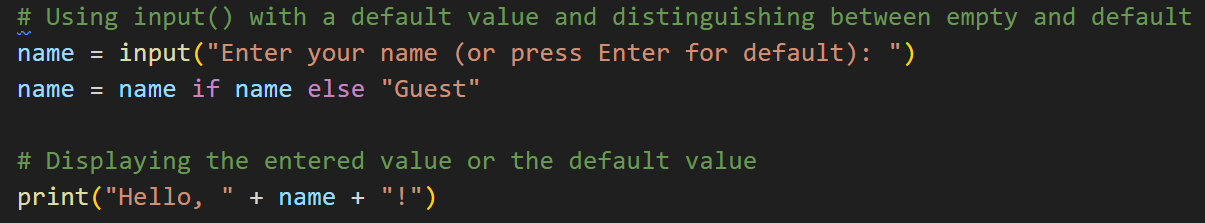
- it return a string

Example:



* You can provide a default value to the input() function by adding a keyword argument default with the desired default value inside the function call.
* If the user presses Enter without typing anything, the default value will be used instead.

Example:



**upper() Method:**

- Purpose: Converts all the characters in a string to uppercase.

**capitalize() Method:**

- Purpose: Capitalizes the first character of a string while converting the rest to lowercase.

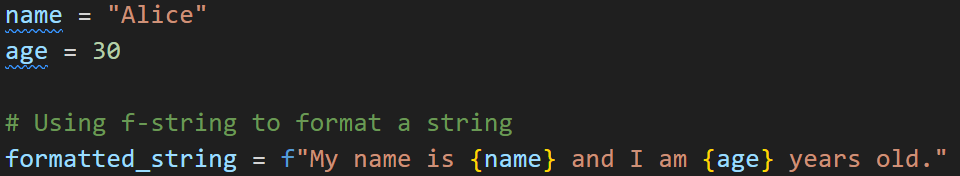
**String**

* In Python, a string is a sequence of characters, and it is one of the basic data types.

Strings can be created using single (') or double (") quotes

Formatted Strings (f-strings)

* In Python, f-strings provide a concise and convenient way to embed expressions inside string literals.
* Inside the f-string, you can include expressions enclosed in curly braces {}.
* These expressions are evaluated at runtime and then formatted into the string.



**Split():**

- Purpose: split the string based on the input (“ “) --> split string when find space

- return : a List

- note : default delimiter (whitespace)

csv\_data = "apple,orange,banana,grape"

fruits = csv\_data.split(',')

**Arithmetic Operations**

**Division (/):**

Example : result = 5 / 3

* This will assign the value 1.6666666666666667 to the variable result.
* In Python 3, the division operator (/) always returns a float.

**Floor Division (//):**

Example : result = 5 // 3

* This will assign the value 1 to the variable result.
* The // operator performs division and rounds down to the nearest integer.

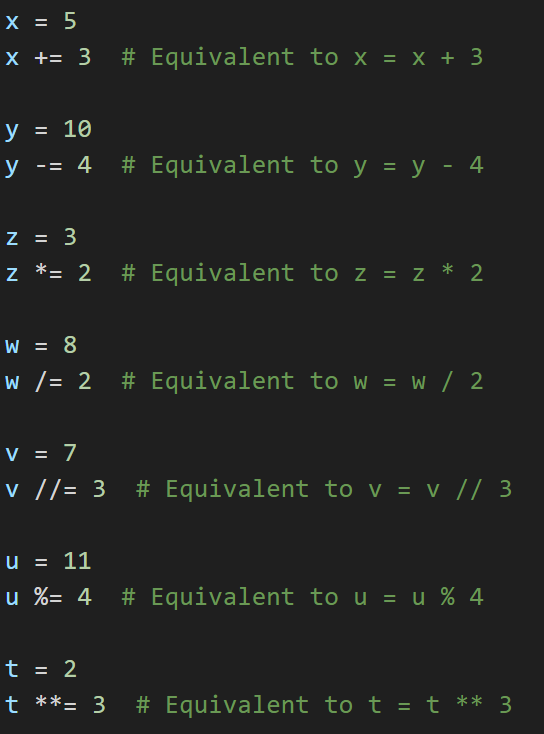
**Exponentiation (\*\*):**

Example : result = 5 \*\* 3

* This will assign the value 125 to the variable result.
* The \*\* operator is used for exponentiation.

**augmented assignment operator**

* They provide a concise way to perform an operation and update the value of a variable in a single step.



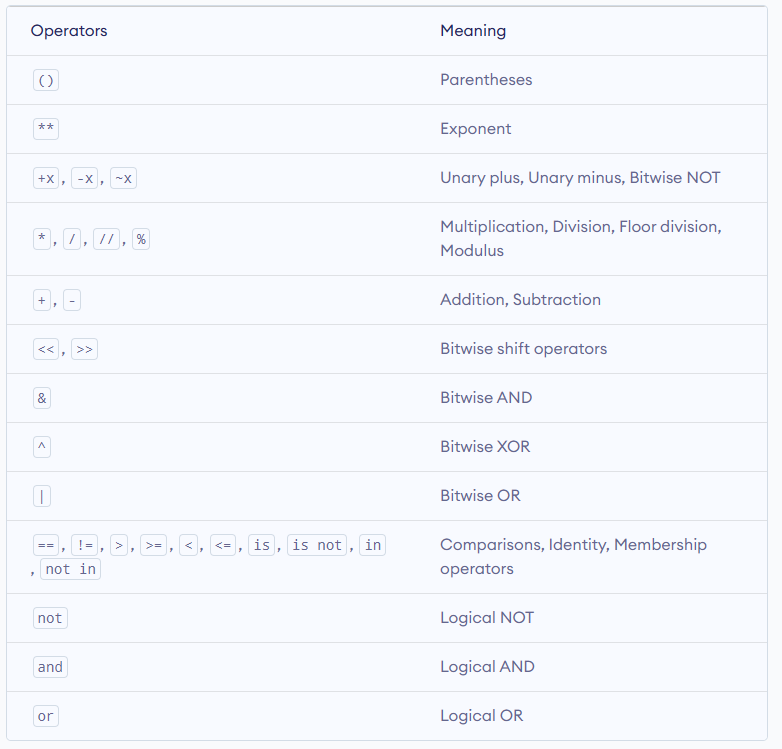
**operator precedence**

* It determines the order in which operations are performed in an expression.

The expression is evaluated based on the priority of operators.

* The operator precedence in Python is listed in the following table.

It is in descending order (upper group has higher precedence than the lower ones).



**Mathematical Functions**

* In Python, you can perform various mathematical operations using built-in functions and operators.

Here are some common mathematical functions and operations:

**1-** **Absolute Value**



**2- Round**

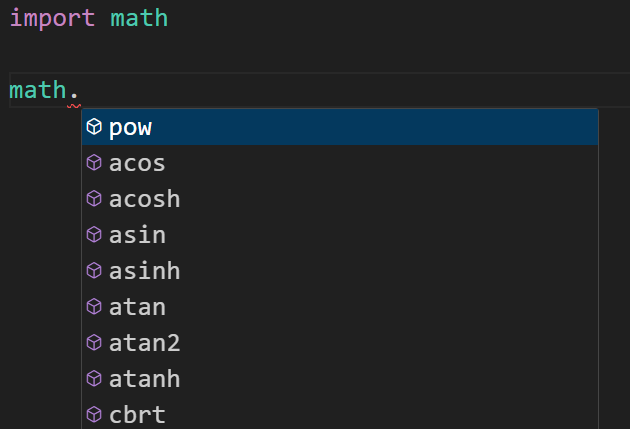
* the function will return the nearest integer.

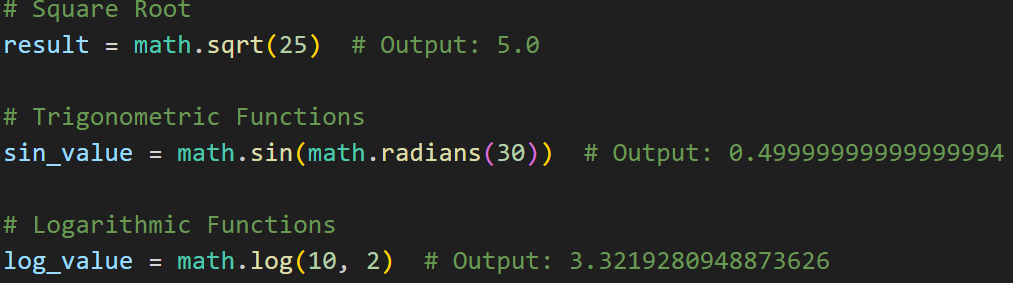


**3- Math Module**

* Python also provides a math module for more advanced mathematical functions:

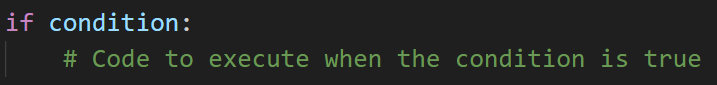
You can import Math Library and type (math.) ,you will see all supported methods



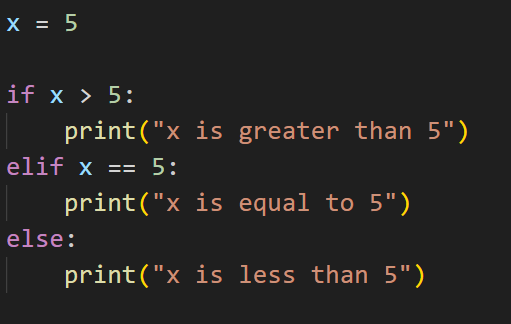


**Python If Statement**

* Python supports the usual logical conditions from mathematics:
* Equals: a == b
* Not Equals: a != b
* Less than: a < b
* Less than or equal to: a <= b
* Greater than: a > b
* Greater than or equal to: a >= b

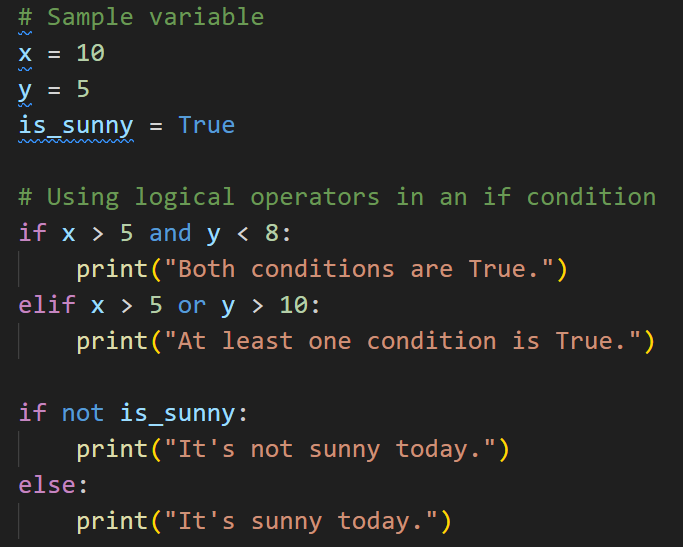


**Example :**



**Logical Operators**

* In Python, logical operators are used to perform logical operations on boolean values.
* These operators allow you to combine or manipulate boolean expressions, resulting in a new boolean value.
* There are three main logical operators in Python: and, or, and not.



**While Loop**

* With the while loop we can execute a set of statements as long as a condition is true.

**Example:**

i = 1  
while i < 6:  
  print(i)  
  i += 1

**1- The break Statement :**

With the break statement we can stop the loop even if the while condition is true:

**Example:**

i = 1  
while i < 6:  
  print(i)  
  if i == 3:  
    break  
  i += 1

**2- The continue Statement :**

With the continue statement we can stop the current iteration, and continue

with the next:

**Example:**

i = 0  
while i < 6:  
  i += 1  
  if i == 3:  
    continue  
  print(i)

**3- The else Statement**

With the else statement we can run a block of code once when the condition

no longer is true:

**Example:**

i = 1  
while i < 6:  
  print(i)  
  i += 1  
else:  
  print("i is no longer less than 6")

## Python For Loops

* A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
* This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

**for variable in iterable:**

**# Code to be executed for each iteration**

**Looping Through a List**

Even strings are iterable objects, they contain a sequence of characters:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  print(x)

apple

banana

cherry

**Looping Through a String**

Even strings are iterable objects, they contain a sequence of characters:

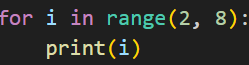
for x in "banana":  
  print(x)

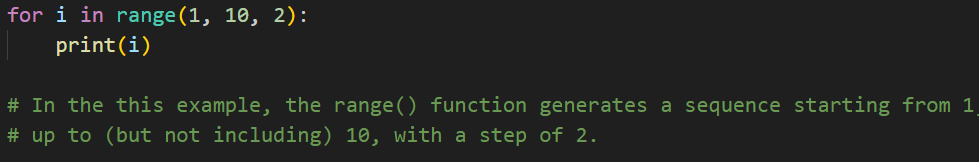
b  
a  
n  
a  
n  
a

**range()** :

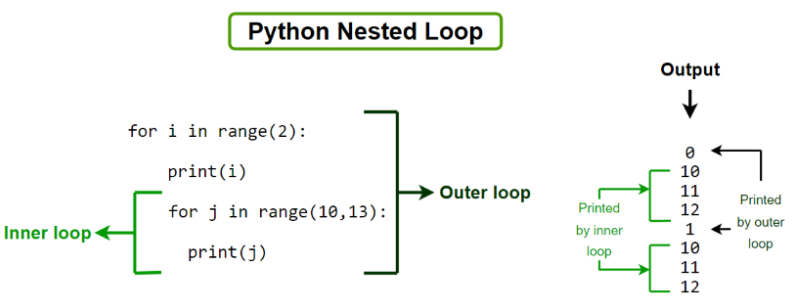
- The range() function in Python is used to generate a sequence of numbers. It's commonly used in for loops to iterate over a sequence of numbers

- It start from index until the last index-1





**Nested for loop**



**List**

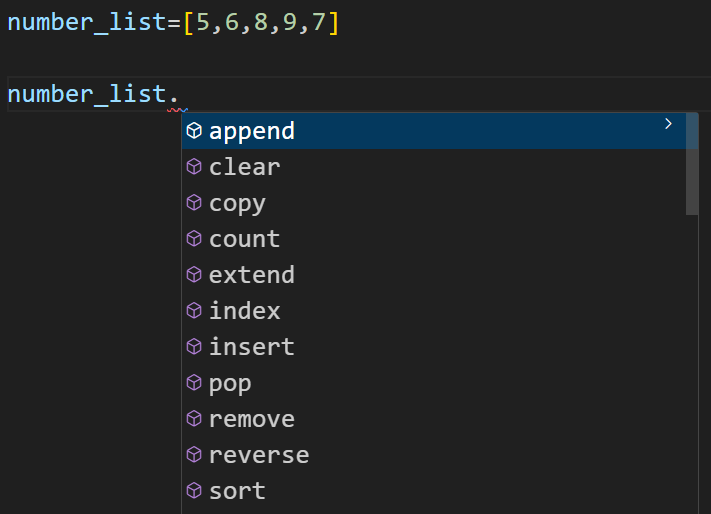
- Python Lists are just like dynamically sized arrays, declared in other languages (vector in C++).

- List items can be of any data type

- A list can contain different data types

list1 = ["abc", 34, True, 40, "male"]

**List methods**



**append()** :

- add new item to the end of the List

**pop()** :

- remove item from the end of the List

**insert()** :

- add new item to the list in specific location

**remove()** :

- remove the first occurrence of item in the List

**clear()** :

- empty the list

**index()** :

- find if a number exist or not (return error) and its first occurrence index

- So its safer to use print(50 in number\_list) .. return false if not found but no error

**count()** :

- find number of occurrence in list

**sort()** :

- sort list in ascending order and has no return

**reverse()** :

- you can use it after sort to descending order your list

**copy()** :

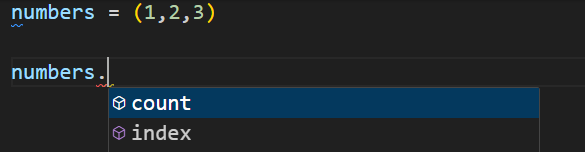
- take a copy from list and return list

## Tuple

* Tuples are used to store multiple items in a single variable.
* Unlike list , we cant add items or remove or modify them

tuple = ("apple", "banana", "cherry")

* We can only get information about it not change it
* Use it when you don’t want anything to change your data



## Unpacking

**# Example 1**

coordinates = (3, 4)

x, y = coordinates

**# Example 2**

s = "hello"

a, b, c, d, e = s

**# Example 3**

person = {'name': 'John', 'age': 30, 'city': 'New York'}

name, age, city = person.values()

## Dictionary

* Dictionaries are used to store data values in key:value pairs.
* A dictionary is a collection which is ordered\*, changeable and do not allow duplicates.
* Dictionary items are presented in key:value pairs, and can be referred to by using the key name.
* The values in dictionary items can be of any data type:

**# Example 1**

thisdict = {  
  "brand": "Ford",  
  "electric": False,  
  "year": 1964,  
  "colors": ["red", "white", "blue"]  
}  
print(thisdict["brand"])

* We can access dic using [] but if we write the wrong name it would give an error

print(customer["Name"])

* So we better use , in case of error it return boolean value ‘None’

print(customer.get("name"))

* we use default value of the name not found

print(customer.get("name",”1”))

# Python Functions

* A function is a block of code which only runs when it is called.
* Function by default return None

def my\_function():  
  print("Hello from a function")  
  
****my\_function()****

**Arguments vs parameters** :

- argument ---> are the actual values that are passed to the function when it is invoked or called

- parameters --> are the variables that are defined or used inside the parentheses while defining a function.

## Keyword Arguments

- You can also send arguments with the key = value syntax.

- This way the order of the arguments does not matter.

- Used to improve readability

========================

def my\_function(child3, child2, child1):  
  print("The youngest child is " + child3)  
  
my\_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")

========================

- Must use keyword argument after positional argument

========================

my\_function("Tobias" , child1 = "Emil")

========================

# Exceptions

* there are several built-in Python exceptions that can be raised when an error occurs during the execution of a program.
* Exceptions: Exceptions are raised when the program is syntactically correct, but the code results in an error

**Here are some of the most common types of exceptions in Python:**

**SyntaxError**: This exception is raised when the interpreter encounters a syntax error in the code, such as a misspelled keyword, a missing colon, or an unbalanced parenthesis.

**TypeError**: This exception is raised when an operation or function is applied to an object of the wrong type, such as adding a string to an integer.

**NameError**: This exception is raised when a variable or function name is not found in the current scope.

**IndexError**: This exception is raised when an index is out of range for a list, tuple, or other sequence types.

**KeyError**: This exception is raised when a key is not found in a dictionary.

**ValueError**: This exception is raised when a function or method is called with an invalid argument or input, such as trying to convert a string to an integer when the string does not represent a valid integer.

**AttributeError**: This exception is raised when an attribute or method is not found on an object, such as trying to access a non-existent attribute of a class instance.

**IOError**: This exception is raised when an I/O operation, such as reading or writing a file, fails due to an input/output error.

**ZeroDivisionError**: This exception is raised when an attempt is made to divide a number by zero.

**ImportError**: This exception is raised when an import statement fails to find or load a module.

# Class

* A Class is like an object constructor, or a "blueprint" for creating objects.

**Naming:**

Capitalize every letter of every word

Class PointOption:

**Object :**

Object in python is dynamic , you can add new attributes to it without the need to modify the class definition

**Constructor** :

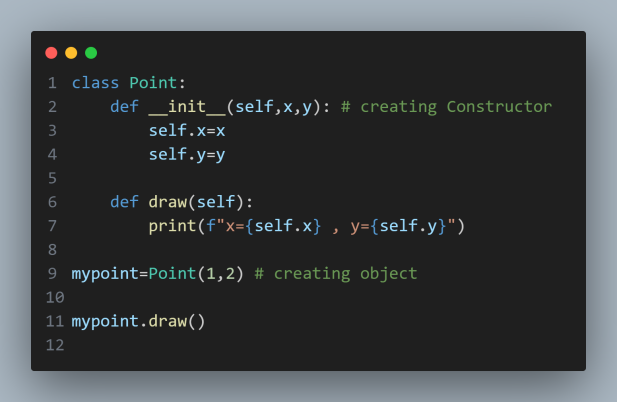
Its magic method and is executed when making new object

slef : its reference to the current object

mypoint = Point(3, 4)

The \_\_init\_\_ method is automatically called with self representing the mypoint instance

class Point: def \_\_init\_\_(self, x, y):



When we call function draw() , python by default gives it current object for self

Like : mypoint.draw(mypoint)

So we don’t have to do that

**Instance attributes vs class attributes**

Instance attributes : they belong to the created object

Class attributes : they belong to any object we create ( like global variables )

**Instance methods vs class methods**

Instance methods : any method created inside a class

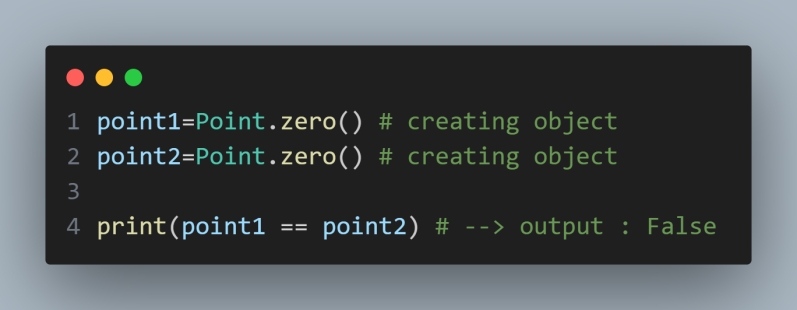
Class methods (factory method): they used to create initialization method that return object

**Magic Methods**

* They called automatically by python interpret when we create an object

1. **\_\_init\_\_** method in Python is used to initialize objects of a class. It is also called a constructor

2- **\_\_str\_\_** used to obtain string representations of objects

**Comparing Objects:**

Comparing gives false because by default (==) compare the addresses for theses objects , so we have to use Magic methods for comparison:

This Magic methods defines the behaviour when comparing objects

\_\_eq\_\_(self, other)

Defines behavior for the equality operator, ==.

\_\_ne\_\_(self, other)

Defines behavior for the inequality operator, !=.

\_\_lt\_\_(self, other)

Defines behavior for the less-than operator, <.

\_\_gt\_\_(self, other)

Defines behavior for the greater-than operator, >.

\_\_le\_\_(self, other)

Defines behavior for the less-than-or-equal-to operator, <=.

\_\_ge\_\_(self, other)

Defines behavior for the greater-than-or-equal-to operator, >=.

Note :

If you define \_\_gt\_\_you don’t have to define \_\_lt\_\_ , python will figure out what to do

**Performing Arithmetic Operations on objects**

We also use magic methods.

\_\_add\_\_(self, other)

Implements addition.

\_\_sub\_\_(self, other)

Implements subtraction.

\_\_mul\_\_(self, other)

Implements multiplication.

\_\_floordiv\_\_(self, other)

Implements integer division using the // operator.

\_\_div\_\_(self, other)

Implements division using the / operator.

\_\_truediv\_\_(self, other)

Implements true division. Note that this only works when from \_\_future\_\_ import division is in effect.

\_\_mod\_\_(self, other)

Implements modulo using the % operator.

\_\_divmod\_\_(self, other)

Implements behavior for long division using the divmod() built in function.

\_\_pow\_\_

Implements behavior for exponents using the \*\* operator.

\_\_lshift\_\_(self, other)

Implements left bitwise shift using the << operator.

\_\_rshift\_\_(self, other)

Implements right bitwise shift using the >> operator.

\_\_and\_\_(self, other)

Implements bitwise and using the & operator.

\_\_or\_\_(self, other)

Implements bitwise or using the | operator.

\_\_xor\_\_(self, other)

Implements bitwise xor using the ^ operator.

# Custom Containers

* To create a custom container in Python, you can define a new class that inherits from an existing container class, such as list or dict.
* You can then add your own methods and attributes to the new class to create a custom container that meets your needs

**The magic behind containers**

\_\_getitem\_\_(self, key)

Defines behavior for when an item is accessed, using the notation self[key]. This is also part of both the mutable and immutable container protocols. It should also raise appropriate exceptions: TypeError if the type of the key is wrong and KeyError if there is no corresponding value for the key.

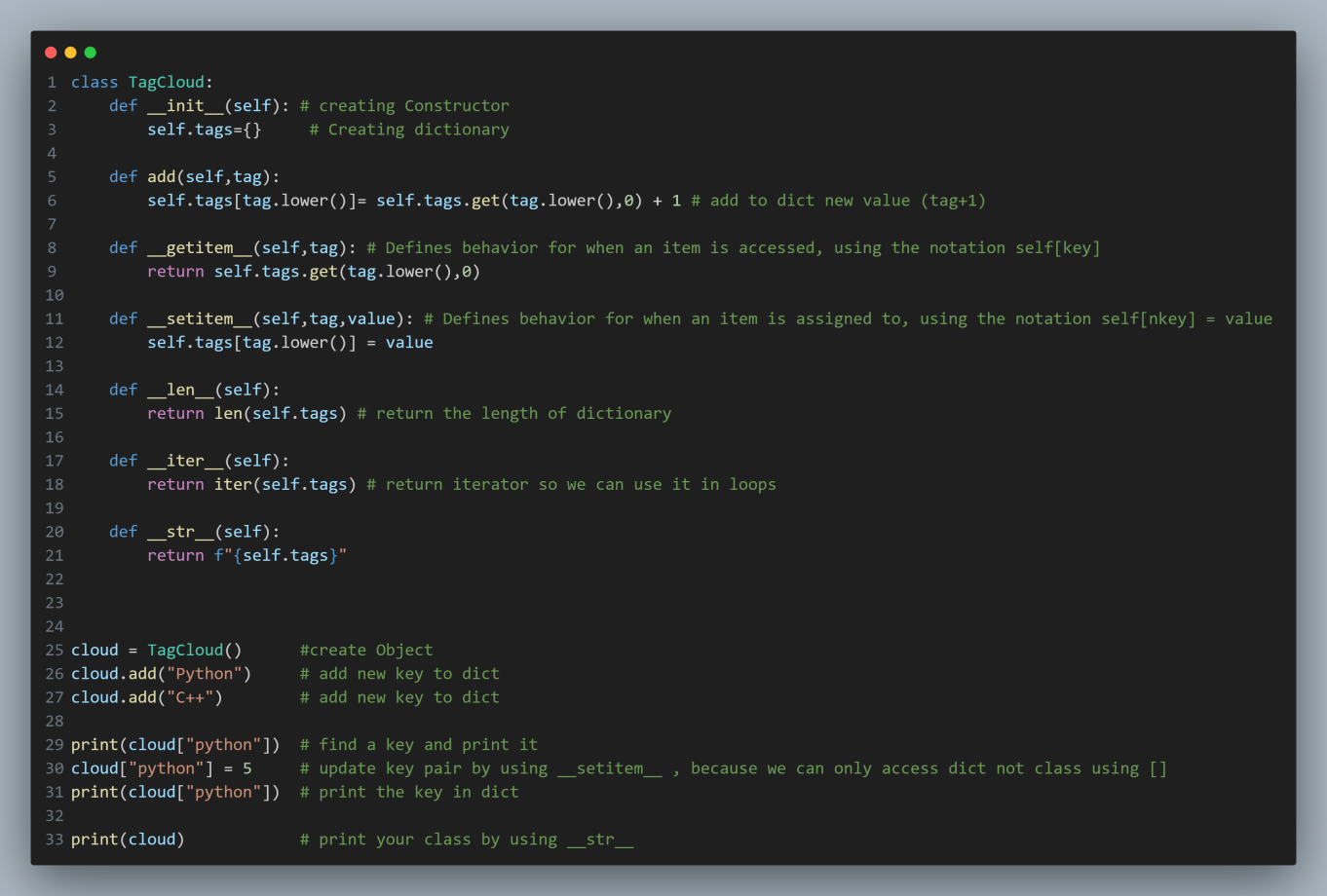
\_\_setitem\_\_(self, key, value)

Defines behavior for when an item is assigned to, using the notation self[nkey] = value. This is part of the mutable container protocol. Again, you should raise KeyError and TypeError where appropriate.

\_\_iter\_\_(self)

Should return an iterator for the container. Iterators are returned in a number of contexts, most notably by the iter() built in function and when a container is looped over using the form for x in container:. Iterators are their own objects, and they also must define an \_\_iter\_\_ method that returns self.

\_\_len\_\_(self)

Returns the length of the container. Part of the protocol for both immutable and mutable containers.

To make our dictionary private we just need to add (\_\_) before its name



This is more like warning (don’t touch its private ) But we still can access it

To know more :

Each class has (\_\_dict\_\_) this contains all the attributes in any class

If we print it we see (\_\_TagCloud\_\_tags) which we can use to see the dict content

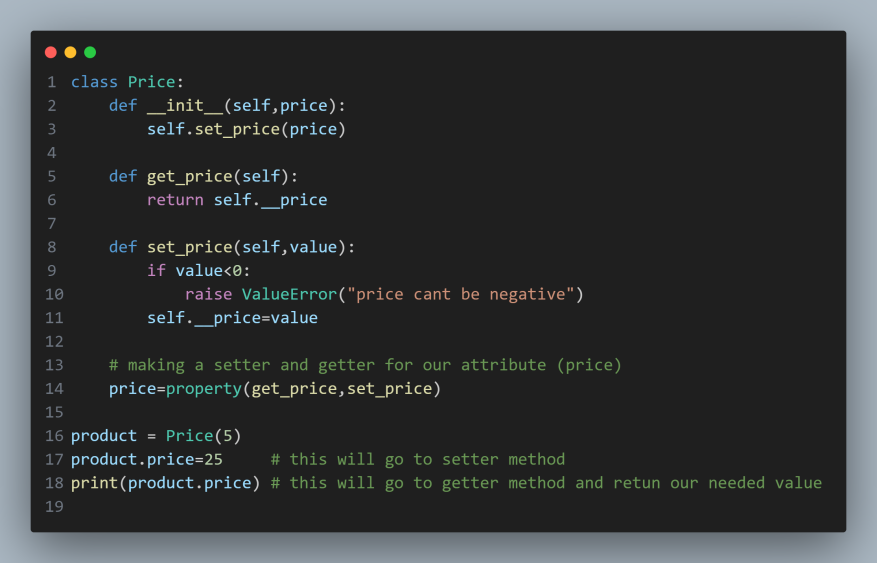
# Properties

**What is property() Function in Python**

* Python property() function is a built-in function that allows us to create a special type of attribute called a property for a class.
* Properties are used to **encapsulate** the access to an object attribute and to add some logic to the process such as computation, access control, or validation.

**Below are the ways by which we can create property for a class in Python:**

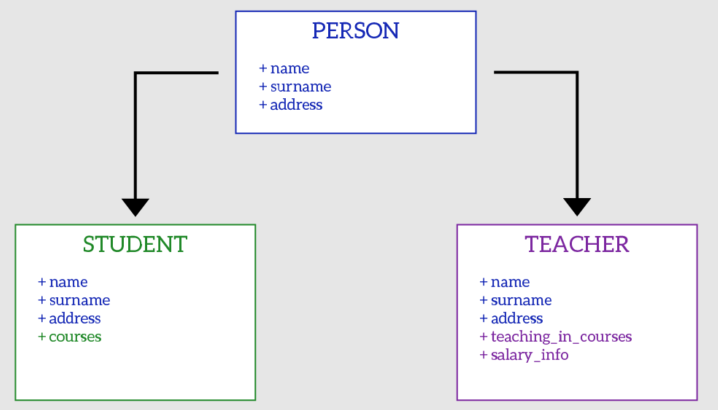
* Using property() method -- we still can access methods (getter and setter)
* Using @property [decorator](https://www.geeksforgeeks.org/decorators-in-python/) -- ( cleaner code and hide getter and setter methods)

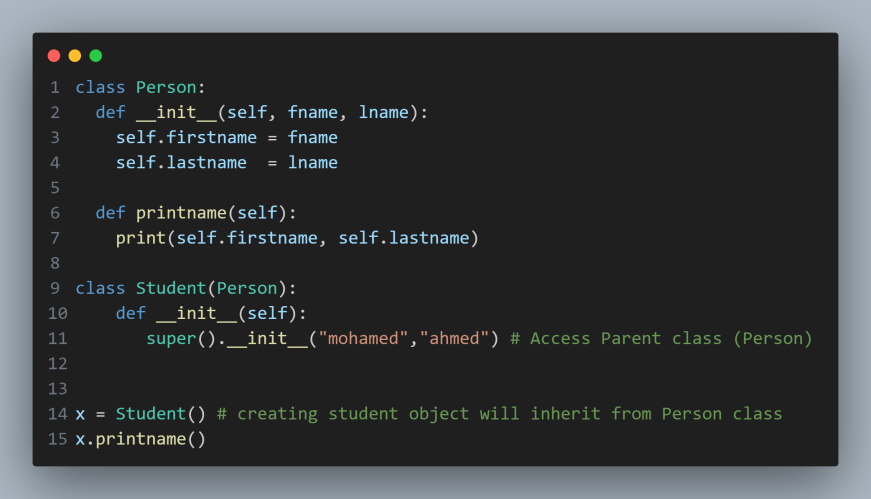


# Inheritance

**What is Inheritance used for**

* we use it when we have repeated code an few classes and we want them to inherit this from one class only (Parent Class) , so that if a problem happened in the code we don’t have to fix it in every class
* We can inherit methods and attributes as well



* We use super() method to access our parent class from child class
* Method Overriding

- init method of student overwrite the init of person class

That’s why we call super.\_\_init\_\_ to call the parent call init

# Object Class

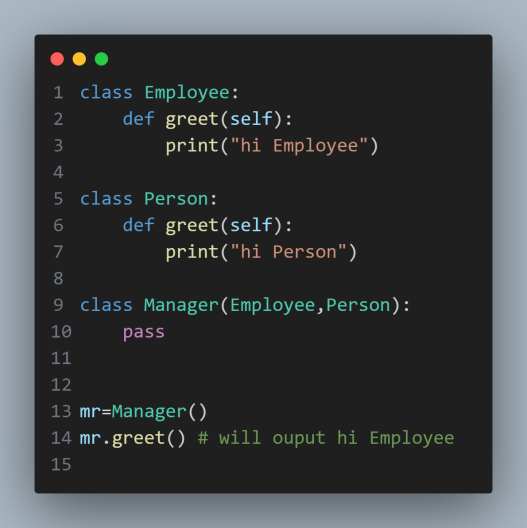
**What is this ?**

* Every class in python derived from class called object
* We can use important method called issubclass(child,parent) to see of any class is derived from another class

# Multiple Inheritance

**What is Inheritance used for**

When we inherit the same method from 2 classes , it runs the first one inherited



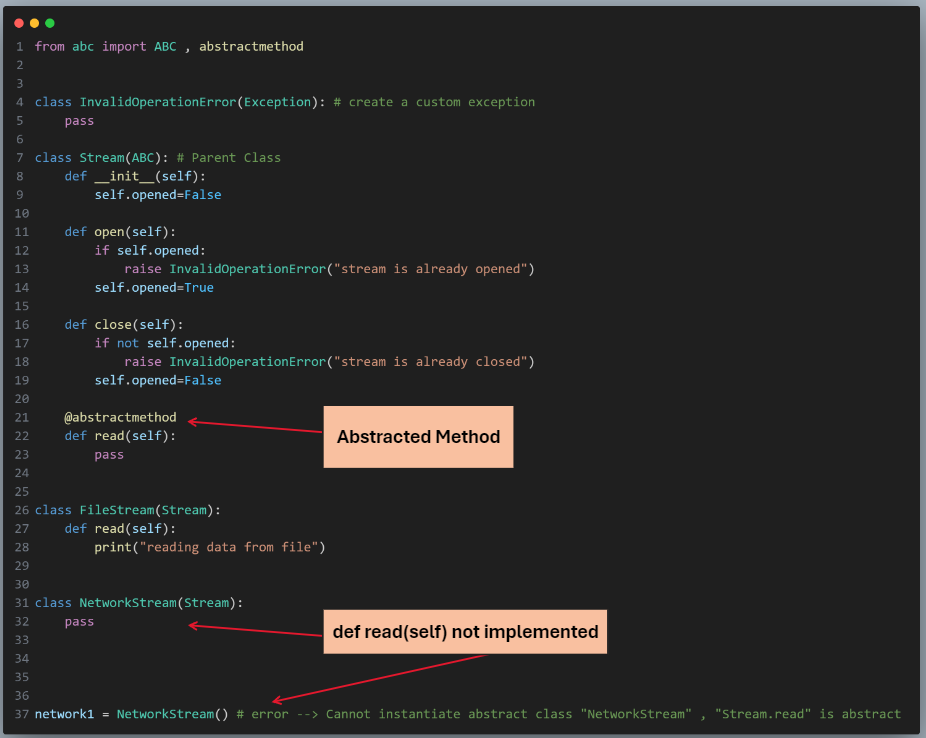
# Python Abstraction Base Classes

**What is it ?**

* Abstraction Base Classes (ABCs) in Python are a way to define abstract classes and abstract methods.
* An abstract class is a class that cannot be instantiated and is meant to be subclassed by non-abstract classes.
* **Abstract methods** within an abstract class are declared but do not provide an implementation.
* **Subclasses** must provide their own implementation of these abstract methods.
* We can’t make instance (object) from abstracted class

**When we use it ?**

* If we have a method in Parent class and want to inherit it in every class Then we should use abstraction Base class and abs decorator
* To tell the interpreter that this method is not implemented in this new class so we don’t forget implementing it



* **This enforce the Subclasses to implement a method**

# Python Polymorphism

**What is it ?**

* The word "Polymorphism" means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

**Function Polymorphism**

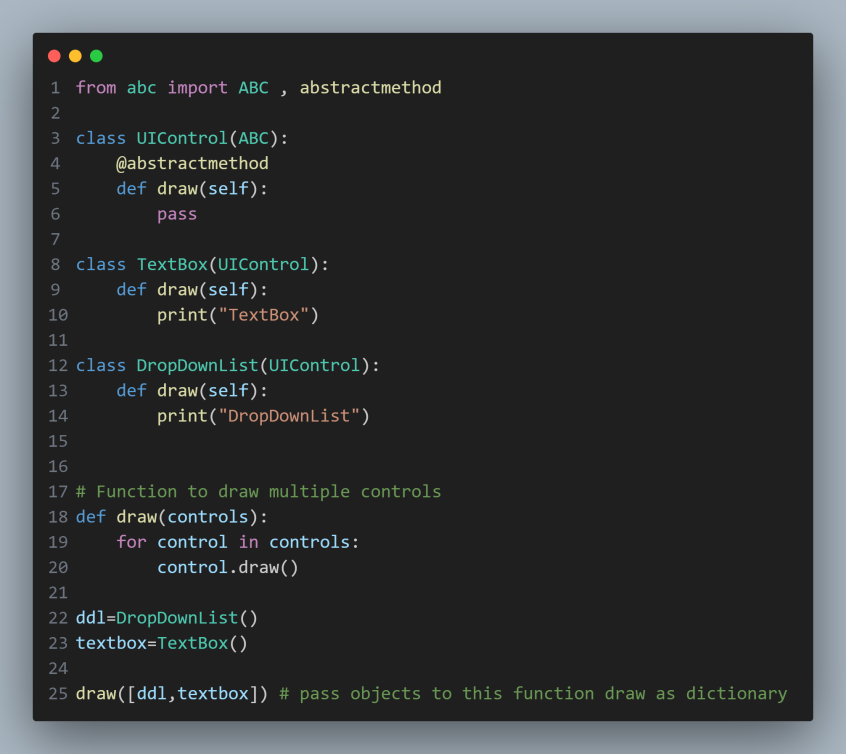
For strings len() returns the number of characters:

For tuples len() returns the number of items in the tuple:

For dictionaries len() returns the number of key/value pairs in the dictionary:

**Class Polymorphism**

Polymorphism is often used in Class methods, where we can have multiple classes with the same method name.

Like draw() method here

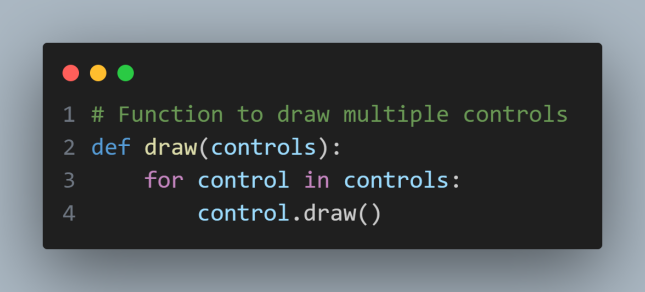
**To achieve Polymorphism behaviour**

1. define a base class
2. In this class define a common behaviour (common method)

# Duck Typing in Python

**What is it ?**

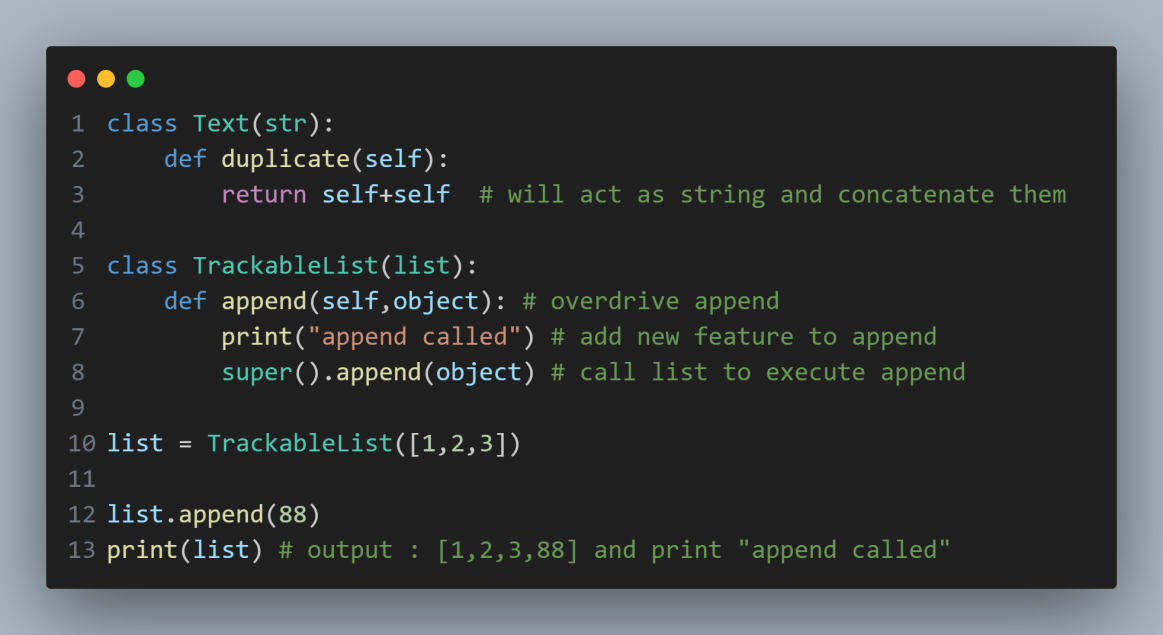
* Duck Typing is a [type system](https://www.geeksforgeeks.org/type-systemsdynamic-typing-static-typing-duck-typing/" \t "https://www.geeksforgeeks.org/duck-typing-in-python/_blank) used in dynamic languages.
* where the type or the class of an object is less important than the method it defines.
* Using Duck Typing, we do not check types at all. Instead, we check for the presence of a given method or attribute.



Like this function draw()

It only looks for the existence of draw() method not the type of control

**Extending Built-in Types**

Example:

# Python Modules

**What is a Module?**

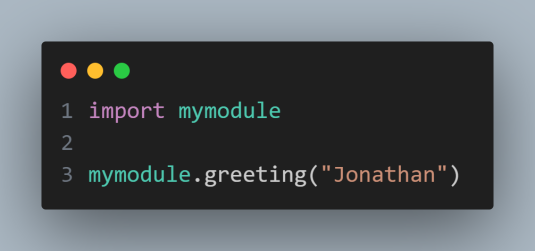
Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

**Create a Module**

To create a module just save the code you want in a file with the file extension .py:

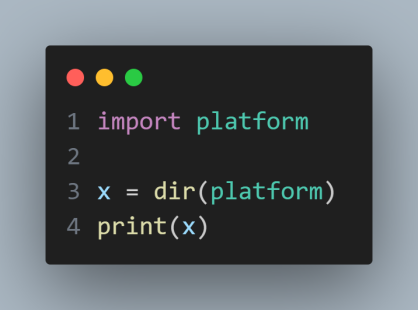
**Use a Module**

Now we can use the module we just created, by using the import statement:

**Re-naming a Module**

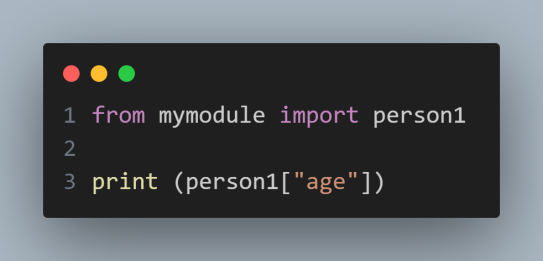
You can create an alias when you import a module, by using the as keyword:

**Using the dir() Function**

There is a built-in function to list all the function names (or variable names) in a module

**Import From Module**

You can choose to import only parts from a module, by using the from keyword.



# Python Packages

**What is a Python Package?**

Package in Python is a folder that contains various modules as files.

**Creating Package**

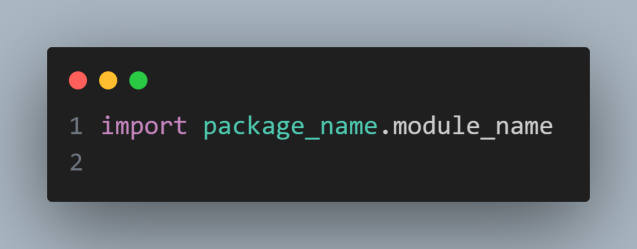
Let’s create a package in Python named mypckg that will contain two modules mod1 and mod2. To create this module follow the below steps:

* Create a folder named mypckg.
* Inside this folder create an empty Python file i.e. \_\_init\_\_.py
* Then create two modules mod1 and mod2 in this folder.

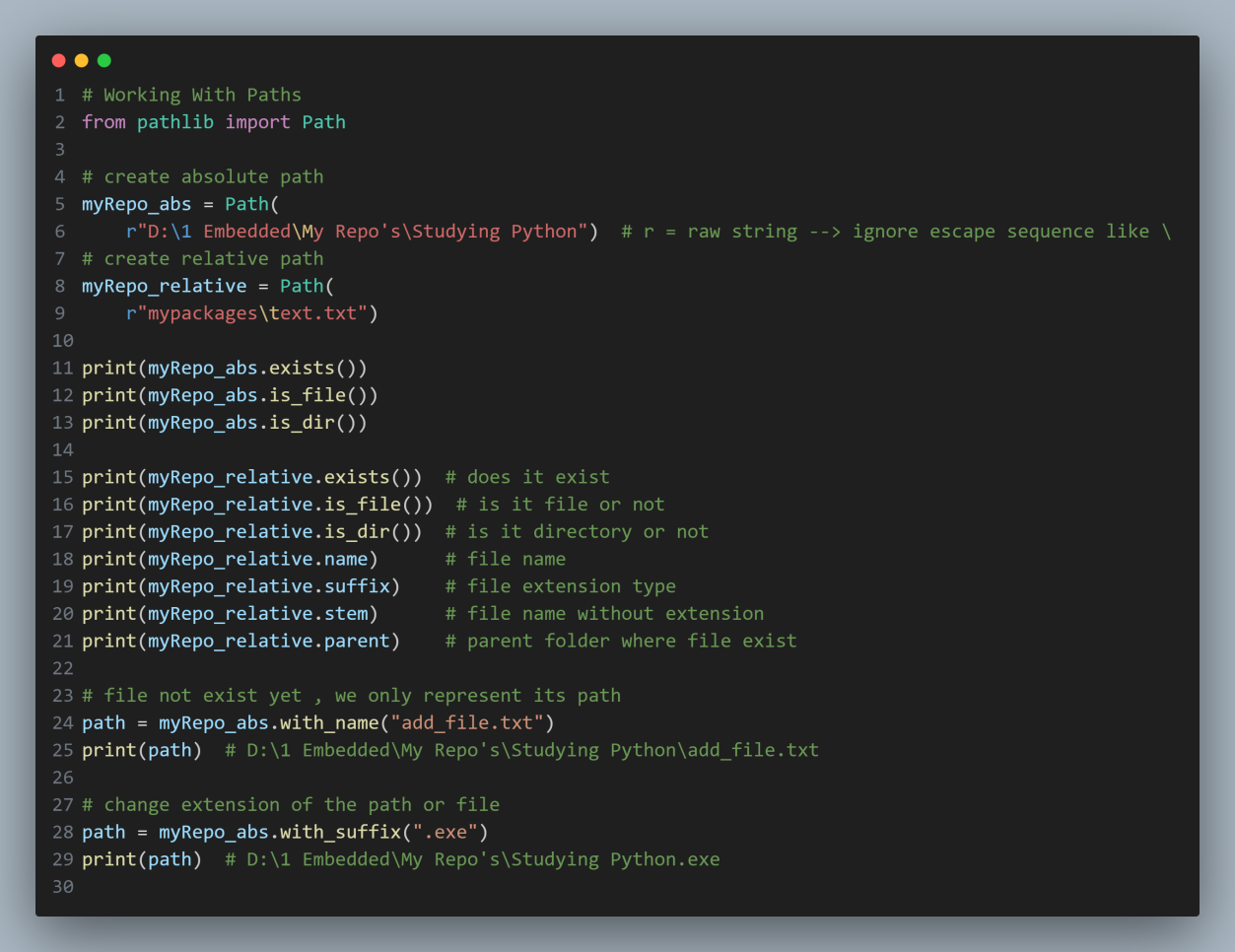
**Understanding \_\_init\_\_.py**

* \_\_init\_\_.py helps the Python interpreter recognize the folder as a package.
* It also specifies the resources to be imported from the modules.
* If the \_\_init\_\_.py is empty this means that all the functions of the modules will be imported.

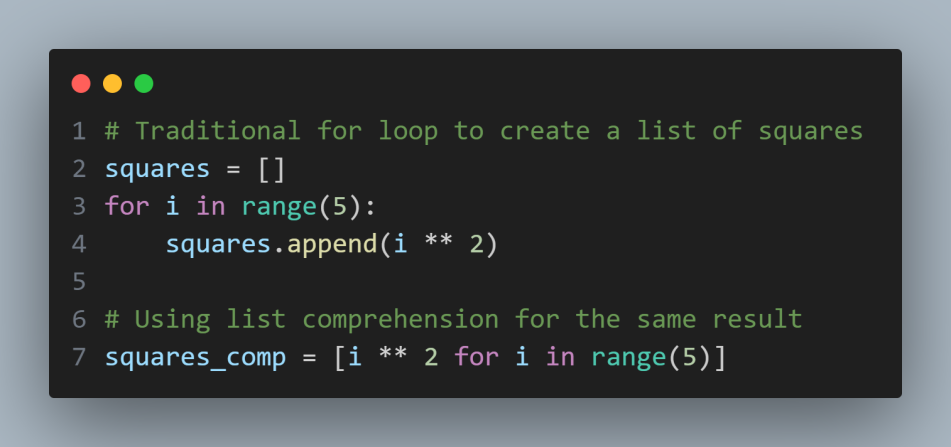
**Import Modules from a Package**

We can import these Python modules using the [from…import statement](https://www.geeksforgeeks.org/import-module-python/) and the dot(.) operator.

# Working With Paths



**List Comprehension**



# Working With Directories

# Working With Files

# Python Package Index

**Pip**

Its used to install other packages, its commands are:

* pip install
* pip list
* pip uninstall

**Virtual Environment**

We use it to install our packages with specific versions sepearted from packages installed in default path , its commands are:

* python -m venv env create env
* env\bin\activate.bat activate enb
* deactivate

**Pipenv**

* Pipenv automatically creates and manages a virtual environment for your projects in another location away from your project, as well as adds and removes dependencies from your Pipfile as you install.
* When we install package using pipenv , 2 files automatically generated to tracks of our files and dependencies

its commands are:

* pipenv shell : Spawns a shell within the virtualenv.
* exit
* pipenv install : will look at pipfile and install all our dependencies
* Pipenv --venv : show the location of env
* pipenv --rm : remove the env

**Pipfile**

It keep tracks of our files and dependencies , so we can run our script on different machine without worrying about missing dependencies

* Pipenv install --ignore--pipfile

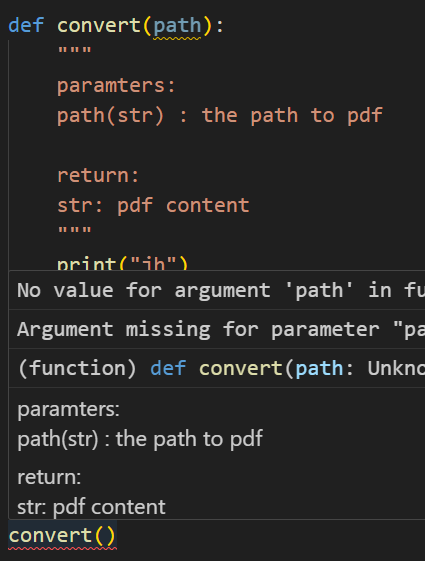
This commands ignore pipfile and install dependencies based on pipfile.lock

Because that’s what we have on our development machine

**Documantion**

**Pip**

You can document your code (class , method) so when someone call them and hover on them a description appears , we call that intellisense



**Pydoc**

You can use this command to print HTML documentation file about package or module

Command :

python -m pydoc -w math

The -w flag is used with pydoc to generate HTML documentation

The -m flag is used to run a module as a script instead of specifying a file path.