Scope of a variable

Variable are either:

- Global
  - Defined outside all functions and classes
  - Or inside functions/classes using the global keyword ▪

Are accessible to any function in the program • Local

  - Defined inside functions and classes only
  - These are only accessible to the function where they are defined.

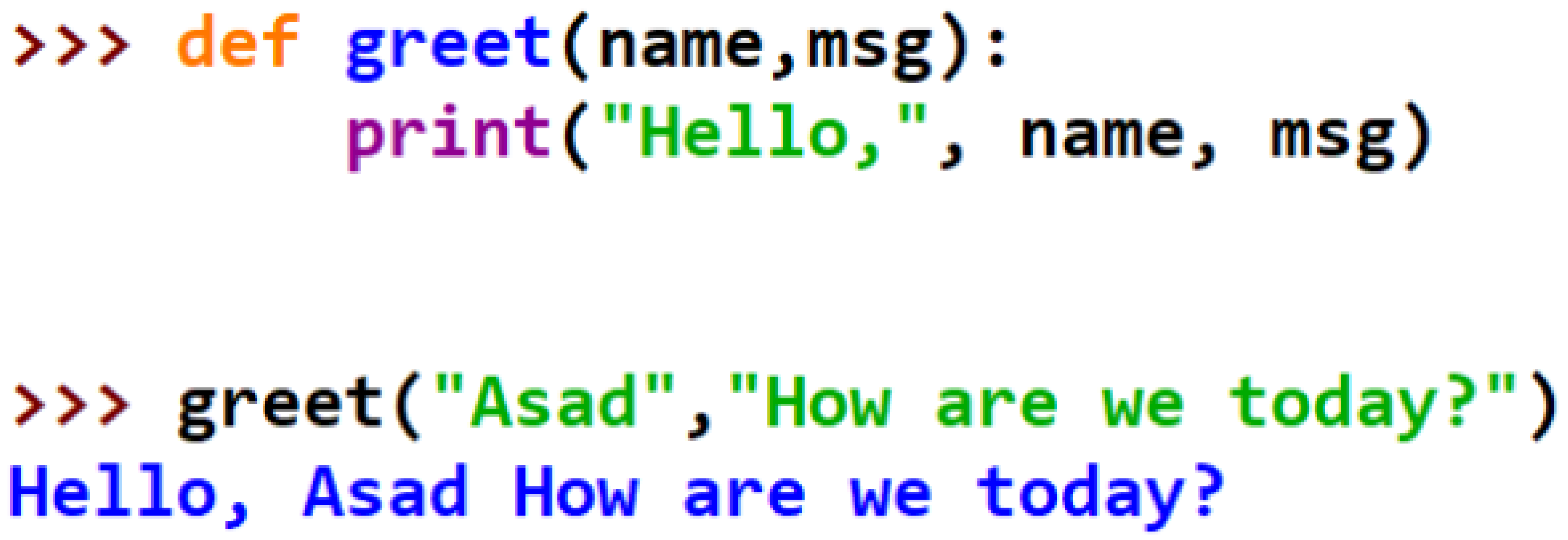
Functions

Types

- Regular Functions: A group of related statements *def function_name(argument1, argument2, …): statement 1*
  nested statement 1.2
- Generators
- Lambdas

<u>Arguments</u>

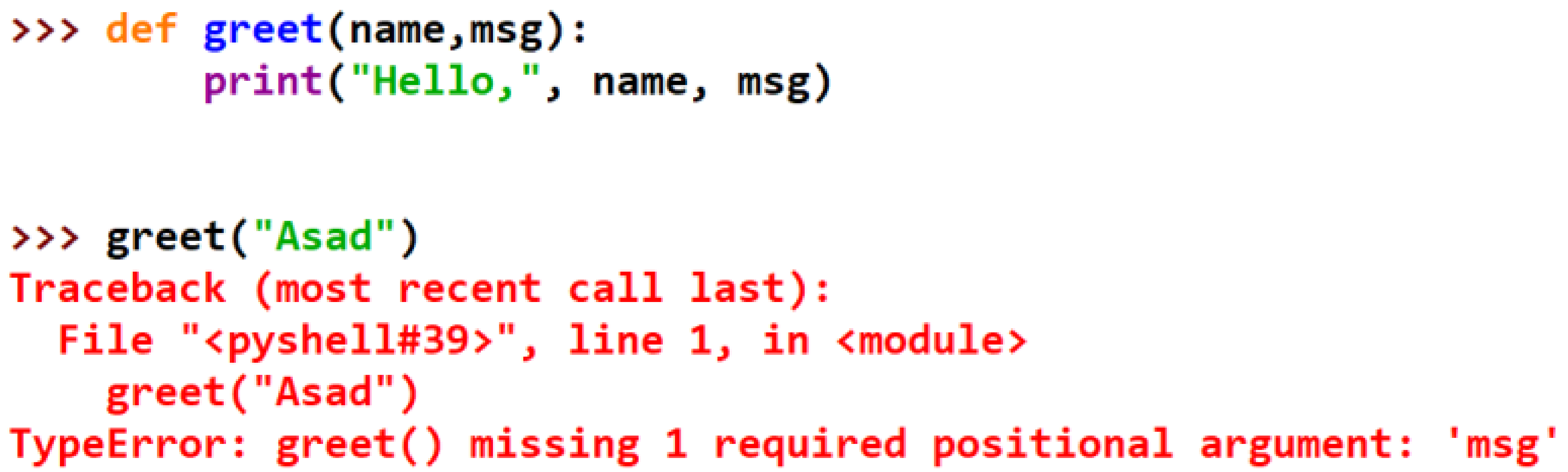
Required Arguments are necessary when functions are called. Here name and msg values must be given when function is called.



```
>>> def greet(name,msg):
        print("Hello,", name, msg)


>>> greet("Asad","How are we today?")
Hello, Asad How are we today?
```

Arguments

Missing required arguments when calling the function gives an error. The example below shows msg argument is missing.



```
>>> def greet(name,msg):
        print("Hello,", name, msg)


>>> greet("Asad")
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    greet("Asad")
TypeError: greet() missing 1 required positional argument: 'msg'
```

Arguments

Default Arguments are filled out when the function is defined.

```
It is so nice to finally meet you!"):
name, ".", msg)
```

d when the function was being

```
ice to finally meet you!
```

msg argument is not defined so the function chooses the default value

Arguments

Default Arguments are overridden if the argument is given when function is called

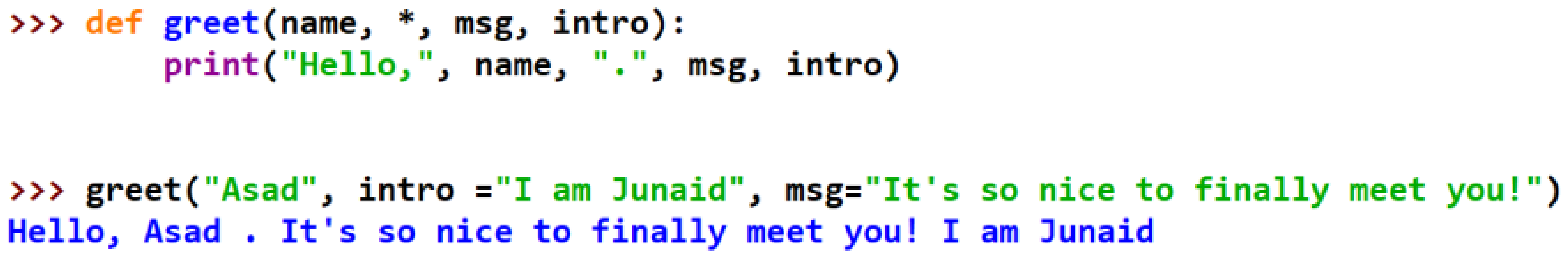
```
sg)


?")
```

msg argument is defined so the function overrides the default value

Arguments

Keyword Arguments require users to state the name of the argument when calling the function

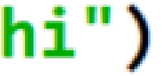
all argument after the asterisk are keyword arguments



```
>>> def greet(name, *, msg, intro):
        print("Hello,", name, ".", msg, intro)


>>> greet("Asad", intro ="I am Junaid", msg="It's so nice to finally meet you!")
Hello, Asad . It's so nice to finally meet you! I am Junaid
```

Arguments

Variable-Length Arguments: Some times the number of arguments receives are unknown.



```
hi")
```

```
hi", "Taimoor")
```

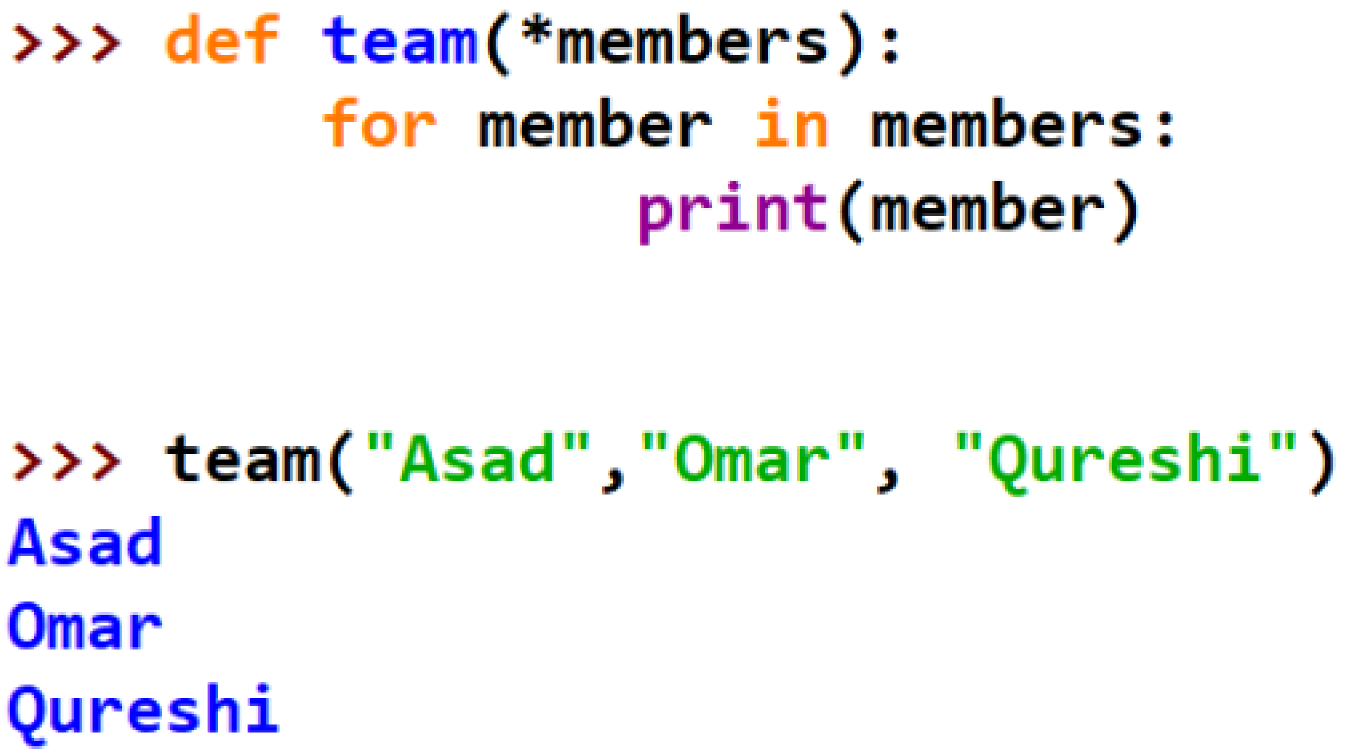
3 arguments only

4 arguments

Arguments

We don’t know the number of input arguments. So we use just one variable when defining the function

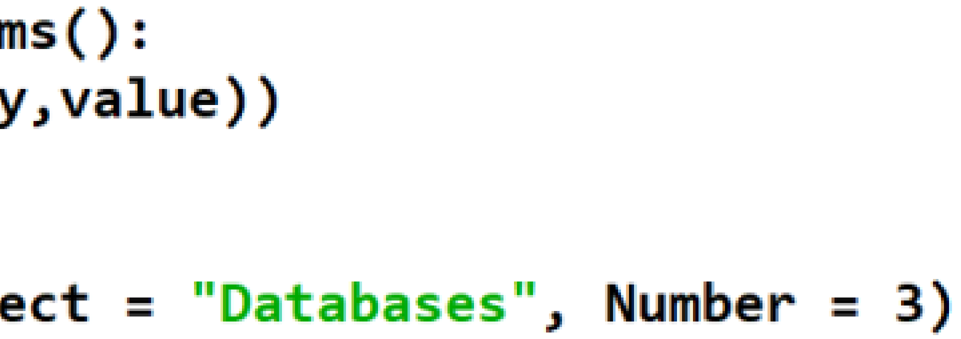
use asterisk right before the single variable



```
>>> def team(*members):
        for member in members:
                print(member)


>>> team("Asad","Omar", "Qureshi")
Asad
Omar
Qureshi
```

Arguments

Variable-Length Keyword Arguments similarly allows any number of inputs as long as input arguments are named use double asterisks when defining keyword based arguments

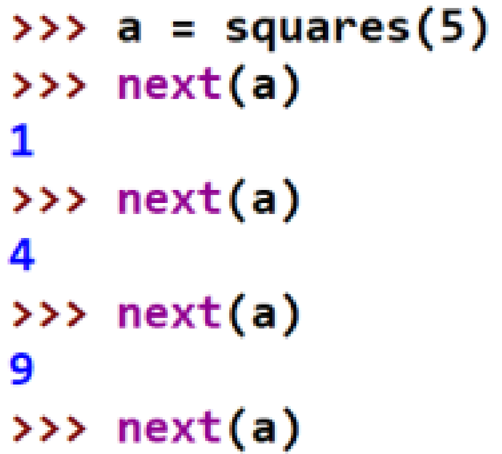


```
ms():
y,value))

ect = "Databases", Number = 3)
```

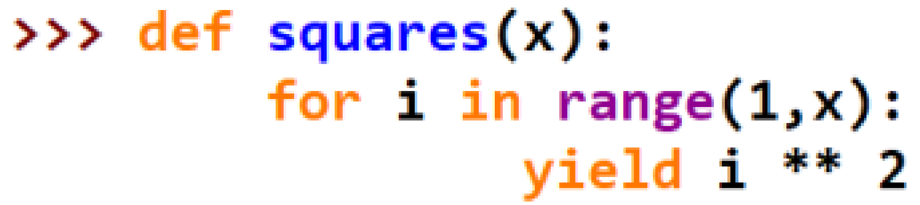
The keyword passed in function are to be named!

Generator functions

Generator functions are actually iterators that yield output of an iteration only when it is needed. It is a function that returns an object (iterator) which we can iterate over (one value at a time).



```
>>> a = squares(5)
>>> next(a)
1
>>> next(a)
4
>>> next(a)
9
>>> next(a)
```



```
>>> def squares(x):
        for i in range(1,x):
            yield i ** 2
```

Generators use the yield keyword instead of return like regular functions

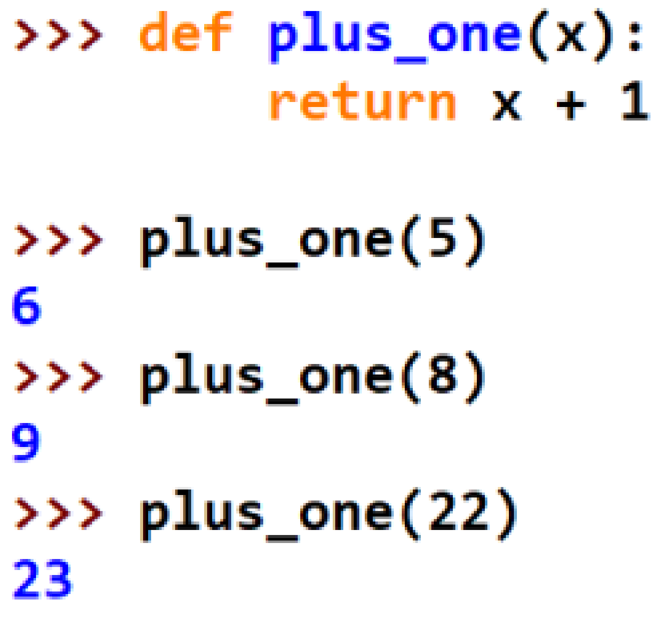
<u>Lambdas</u>

Best for single line <u>anonymous</u> functions Can be used where *def* cannot

```
AB80>
```

<u>x</u> is an argument NOT a name.
Moreover, you don’t use the return word here.

Lambdas(Activity)



```
>>> def plus_one(x):
        return x + 1

>>> plus_one(5)
6
>>> plus_one(8)
9
>>> plus_one(22)
23
```

THE SAME AS

Write a lambda function that returns True only if the number is even. Otherwise False

Classes

- Help in making code reusable
- Keep the data members and methods together in one place •

Inheritance

- Groups related functions together to improve program readability.

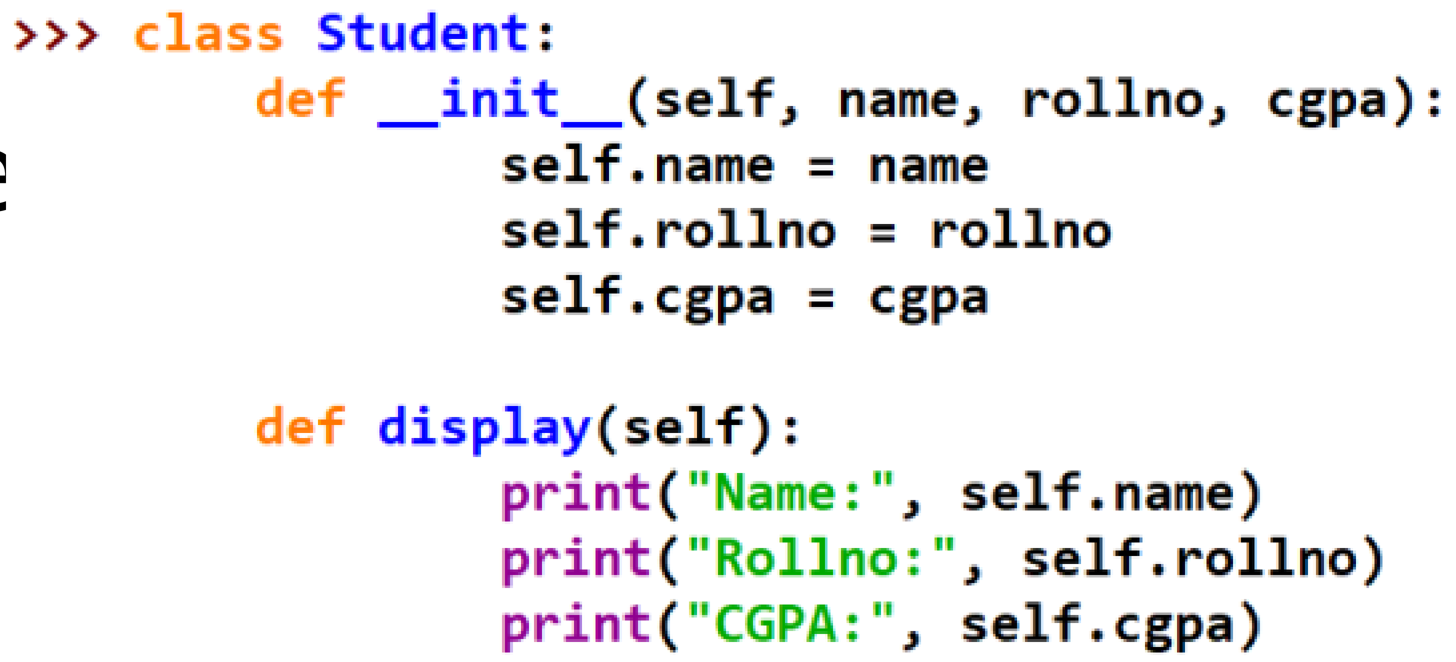
Classes: What is the syntax?

```
class name:
        "documentation"
         statements
-or
class name(base1, base2, ...): # inheritance
..........................................................
Most, statements are method definitions:
def name(self, arg1, arg2, ...):
..........................................................
```

May also be *class variable* assignments

Classes

Example



```
>>> class Student:
        def __init__(self, name, rollno, cgpa):
                self.name = name
                self.rollno = rollno
                self.cgpa = cgpa

        def display(self):
                print("Name:", self.name)
                print("Rollno:", self.rollno)
                print("CGPA:", self.cgpa)
```

# constructor

_ _ init _ _ is the class constructor.

UNDERSCORES

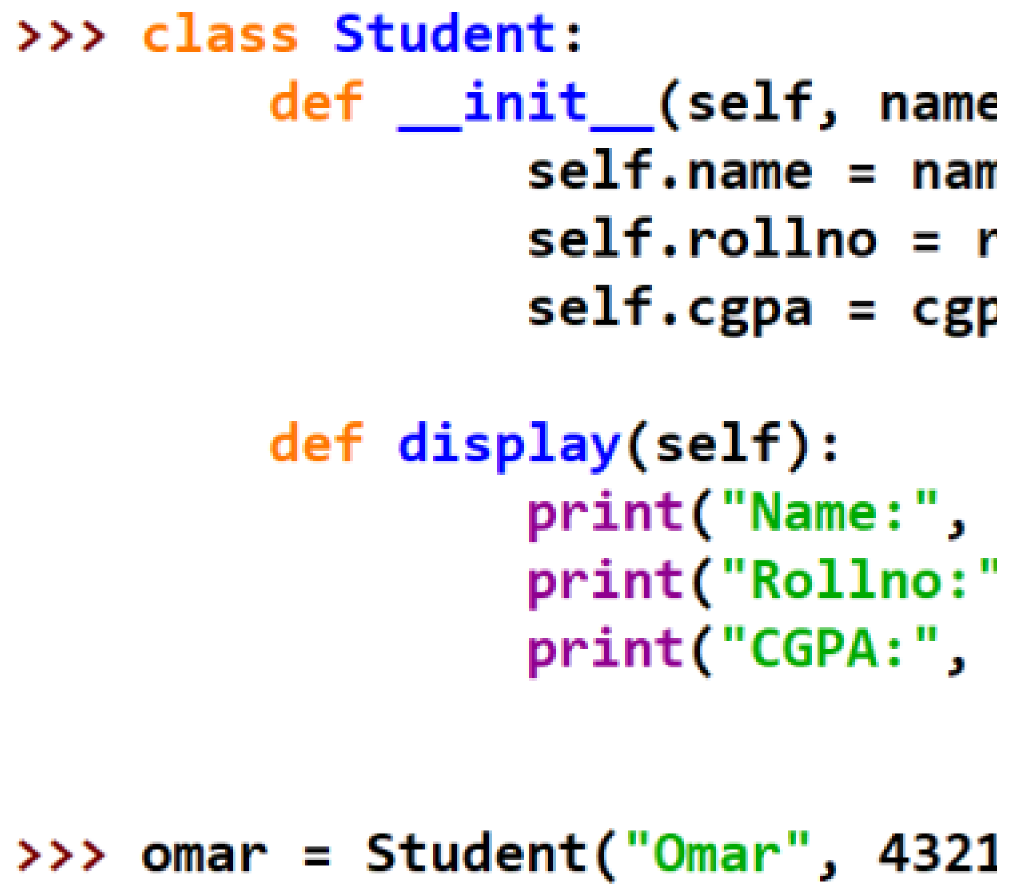
.

# method

USE DOUBLE

Classes

Creating an instance simply requires calling the class object



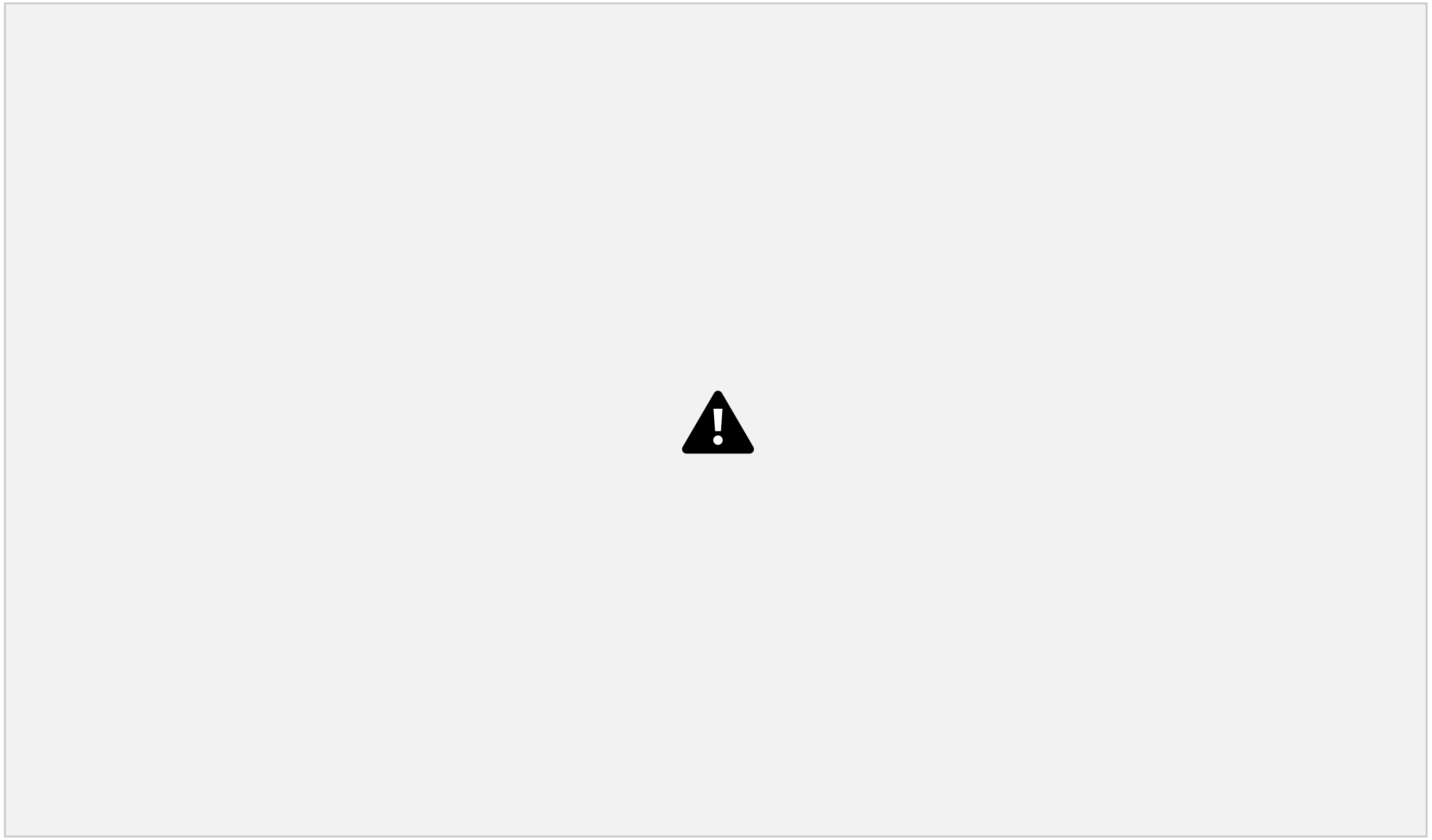
```
>>> class Student:
        def __init__(self, name
                self.name = nam
                self.rollno = r
                self.cgpa = cgp

        def display(self):
                print("Name:",
                print("Rollno:"
                print("CGPA:",


>>> omar = Student("Omar", 4321
```

What is <u>se</u>

The **<u>self</u>** acts just like the **<u>this</u>**
pointer in C++. and
references the object being
talked about. The difference
is that it must be explicitly
defined.

<u>Classes</u>

Dot Notation is used for using class methods



Classes

We also use dot notation for inspecting instance variables

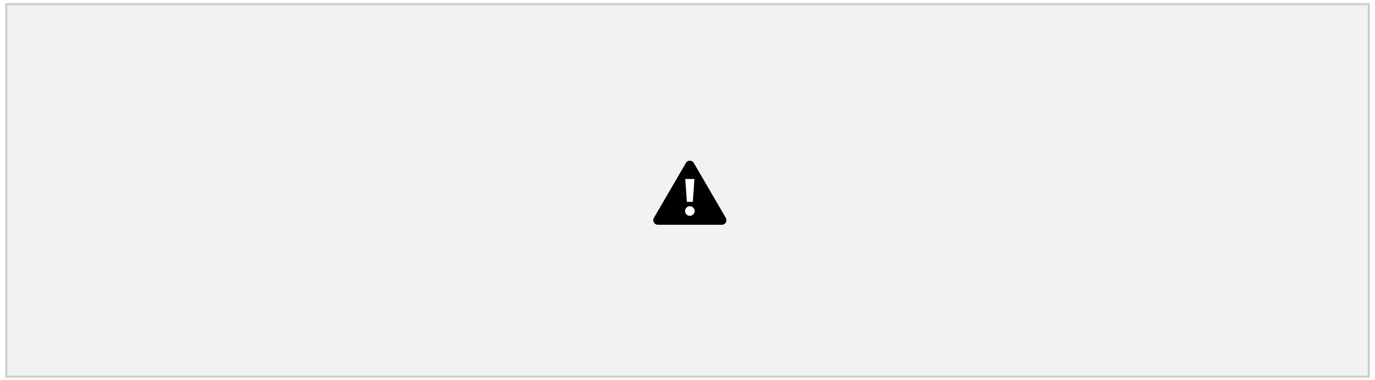
Classes (Activity)

Create a Circle class and initialize it with radius. Make two methods getArea and getCircumference inside this class.

Modules

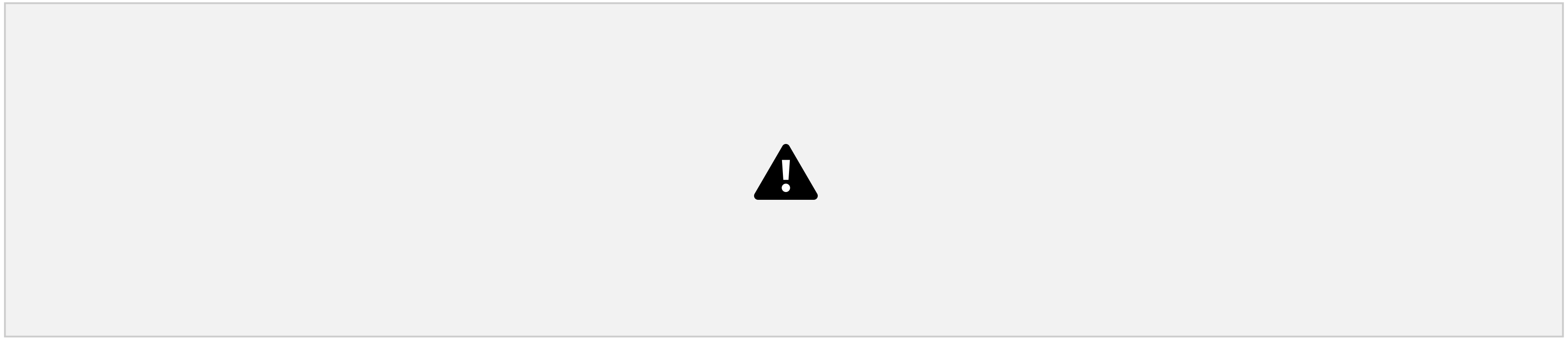
Modules are simply Python Files containing Python Code

You can import functions, classes or variables and use them in the present file you're working on…



Modules

You can create your own modules too



We don’t use .py extension when importing modules. Just the filename.

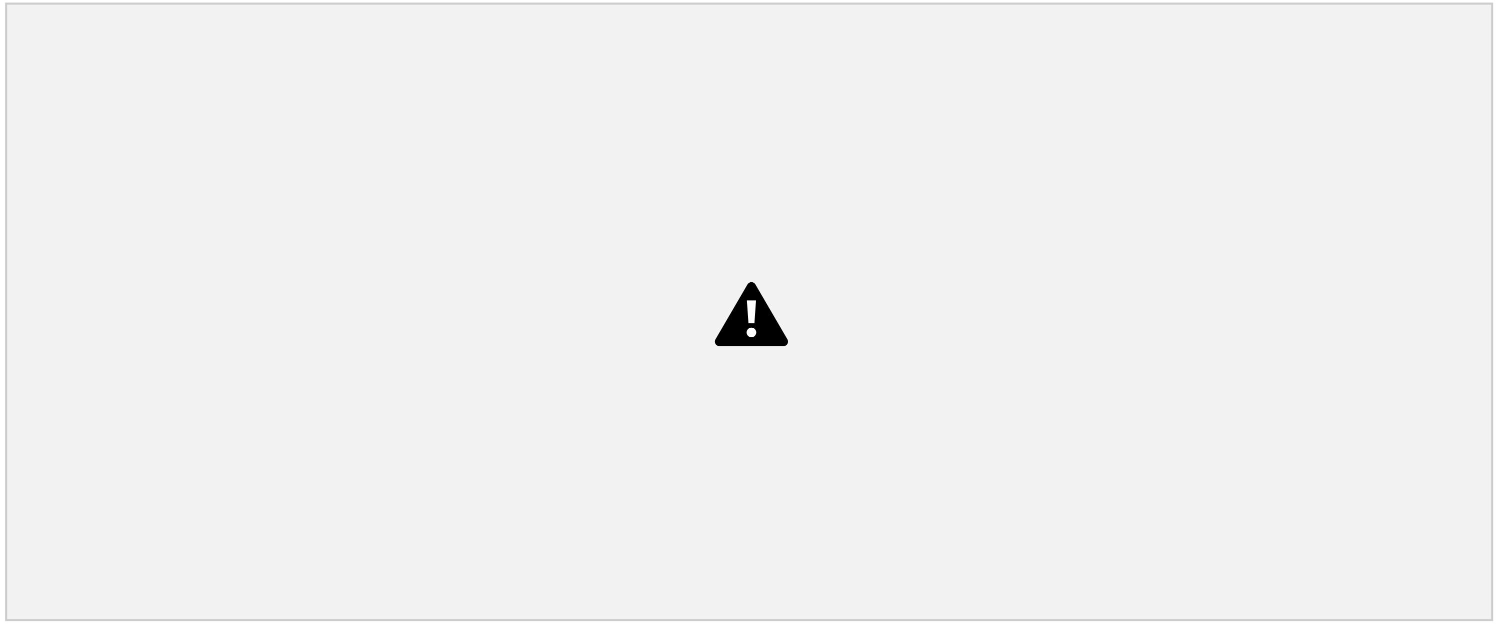
Modules

You can rename modules Use the “as” keyword to rename when importing modules.

Try-catch Statements

Used for Error-handling

or override Python’s default behavior for error



Try-catch Statements

We can manually raise exceptions too. We can also write what message to display when raising exceptions.