Lesson 1: Week 4 - Modules and Recursion

Created: 2024-05-27T01:25:37.937984+10:00

Creating modules

You might find yourself creating your own functions or classes (to be discussed later) that you would like to use in various programs. Rather than copying and pasting their definitions into each program it is far better to put them into a module, and then import that module whenever you need to. Just as you can put code into functions, you can put functions (and classes) into modules.You can create your own modules by writing Python code and saving it in a file with a ".py" extension. Then you can import these modules into your code whenever you need them.For example, suppose you have a module called "hello.py" which contains the following code:def say_hello(): print('Hello world!')You can import and use this module just like any other module (do not include the ".py" part of the file name):from hello import say_hello say_hello()Importing quietlySometimes you might have code in your module that is outside of a function or class. For example, suppose the contents of "hello.py" are this:def say_hello(): print('Hello world!') print("Python is fun!")When you import this module into your program, the Python interpreter executes the code in the module, including line 3. This will generate output in your program that you probably don't want:Python is fun! Hello world! Python is fun! is a consequence of line 3 of your module, which was executed when it was imported.It is better if your module is imported quietly. You make sure this happens by modifying your module as follows:def say_hello(): print('Hello world!') if __name__ == '__main__': print("Python is fun!")When a file executes it contains is a special variable __name__ that indicates whether the file has been executed as a standalone file or as an imported module. In the former case, the value of this variable is '__main__', otherwise it is the name of the module, 'hello'. This gives you a way of checking whether your module was executed as a standalone file or as an imported module, and modifying its behaviour accordingly. In the code above, line 4 does the check, and print("Python is fun!") is considered only if the module is being executed as a standalone file. When you import the module, it will not be executed.

Modules Example

Recursion

You can get a function to call itself. This is a very powerful technique known as recursion. A function that calls itself is said to be a recursive function.Suppose you want to create a function that calculates, for a given number n, the factorial of n, which is 1 × 2 × 3 × ... × n.For example, the factorial of 6 is 1 × 2 × 3 × 4 × 5 × 6 = 720.You can do this without recursion, using a loop: def factorial(n): result = 1 for x in range(1, n+1): result = result * x return result number = input('Enter a number: ') number = int(number) print(f'The factorial of {number} is {factorial(number)}.')Alternatively, you can do it with recursion.The trick is to notice that the factorial of n can be obtained by multiplying n by the factorial of (n-1).Hence, we can do this:def factorial(n): if n== 0 or n == 1: return 1 else: return n * factorial(n-1) # Recursion - the function calls itself number = input('Enter a number: ') number = int(number) print(f'The factorial of {number} is {factorial(number)}.')You need to be careful, when defining a recursive function, that it does not keep calling itself forever. That is why the recursive function above checks the value of n and decides what to do accordingly. There are two cases:n = 1. If n is 1 then the function doesn't call itself - it just returns 1. This is sometimes called the base case.Otherwise. If n is not 1 then the function calls itself. This is sometimes called the recursive case.The base case is important - it stops the function from calling itself forever. Notice what happens without it:def factorial(n): return n * factorial(n-1) number = input('Enter a number: ') number = int(number) print(f'The factorial of {number} is {factorial(number)}.')It is very common for a recursive function to distinguish these two cases - the base case and the recursive case. When you define a recursive function, check that you have your cases covered.

More Recursive Examples

Adding Two Numbersdef add(a,b): if b == 0: return a return add(a,b-1) + 1 print(add(4,9))Multiply by 4def mult4(n): if n == 1: return 4 return mult4(n-1) + 4 print(mult4(1)) print(mult4(5))Factorial - Better Versiondef fact(n): if n

Fibonacci Iterative, Recursive, and Memoise Versions

Further reading

You might find the following helpful:The Python Tutorial at w3schools.com