Chapter_6_Fruitful_functions

March 6, 2024

The print statements we wrote are useful for debugging, but once you get the function working, you should remove them. Code like that is called scaffolding because it is helpful for building the program but is not part of the final product.

As you should expect by now, you can call one function from within another

What happens if we call factorial and give it 1.5 as an argument?
factorial(1.5)
RuntimeError: Maximum recursion depth exceeded It looks like an infinite recursion. How can that be ? The function has a base case—when n== 0. But if n is not an integer, we can miss the base case and recurse forever

We can try to generalize the factorial function to work with floating-point numbers, or we can make factorial check the type of its argument. The first option is called the gamma function and it's a little beyond the scope of this book. So we'll go for the second

[1]:
```
# if not isinstance(n, int):
```

[2]:
```
def factorial(n):
    if not isinstance(n, int):
        print('Factorial is only defined for integers.')
        return None
    elif n < 0:
        print('Factorial is not defined for negative integers.')
        return None
    elif n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

This program demonstrates a pattern sometimes called a guardian. The first two conditionals act as guardians, protecting the code that follows from values that might cause an error. The guardians make it possible to prove the correctness of the code.

[ ]:
```

```