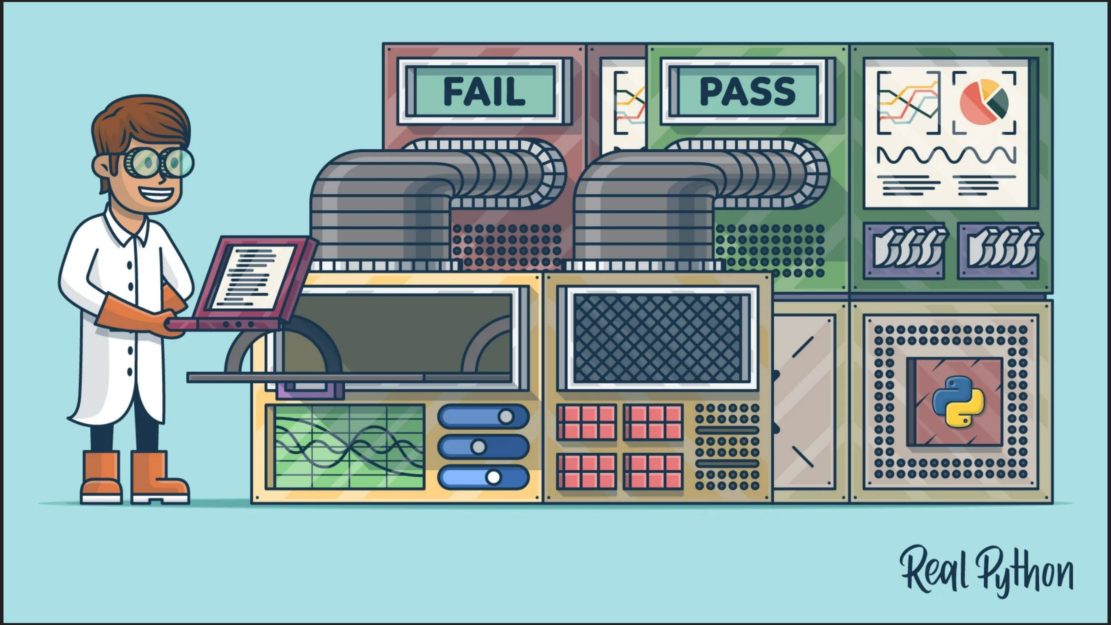
Functions and Testing



Why do we care?

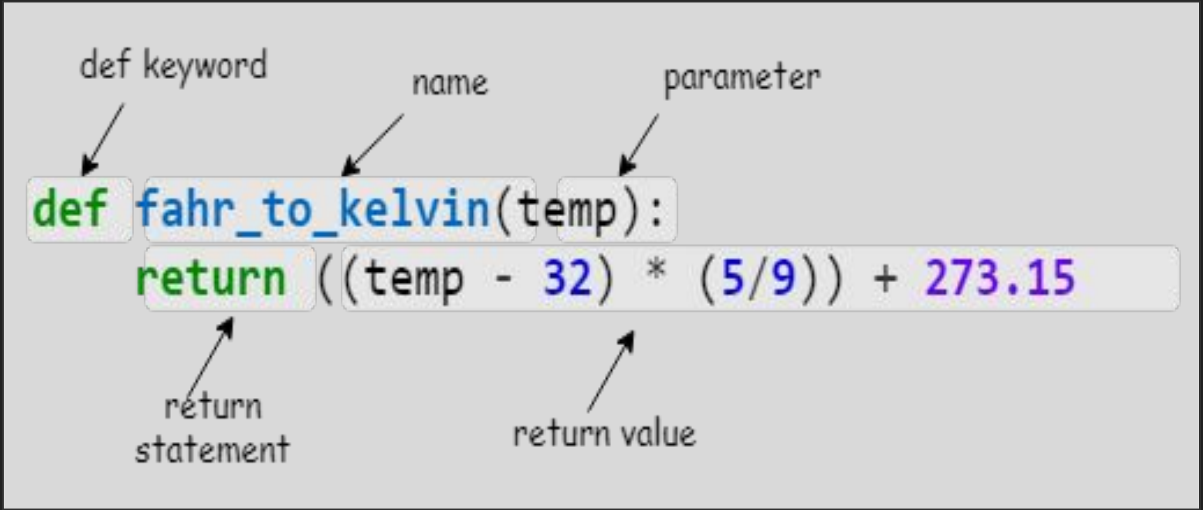
Functions

- Reusability
  - Repeated code can avoided
- Organization
  - Allows code to be broken into more readable chunks
- Abstraction
  - We only need to care about inputs and outputs
  - Most people don’t care how the *sin* function works!

Testing

- Validation
  - Fastest way to check if your code is doing what you think it’s doing
- Understanding
  - Forces you to think critically about the problem you are solving
- Debugging
  - Directs you to where your program might be failing

Functions in Python



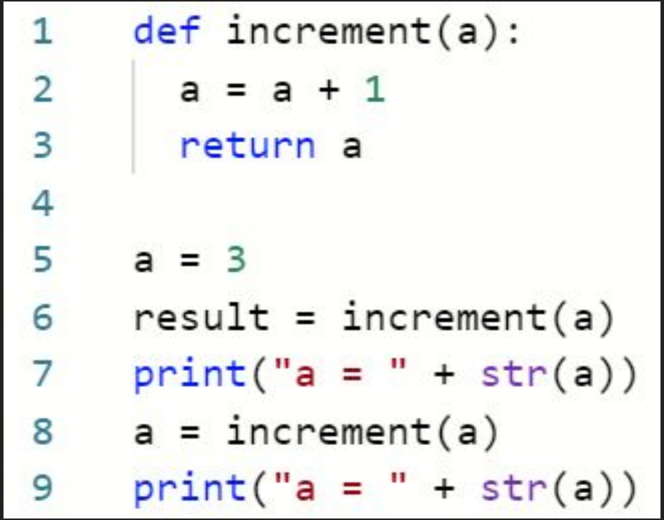
```
def fahr_to_kelvin(temp):
    return ((temp - 32) * (5/9)) + 273.15
```

fahr_to_kelvin(32) ⇒ 273.15

fahr_to_kelvin(41) ⇒ 278.15

Function Scope

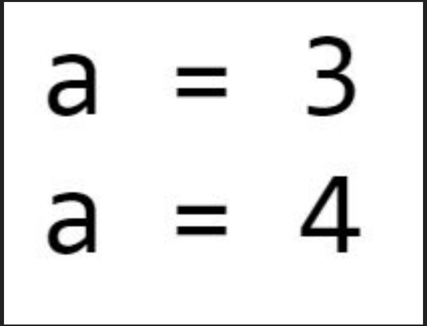
Code:



```
def increment(a):
  a = a + 1
  return a

a = 3
result = increment(a)
print("a = " + str(a))
a = increment(a)
print("a = " + str(a))
```

Output:

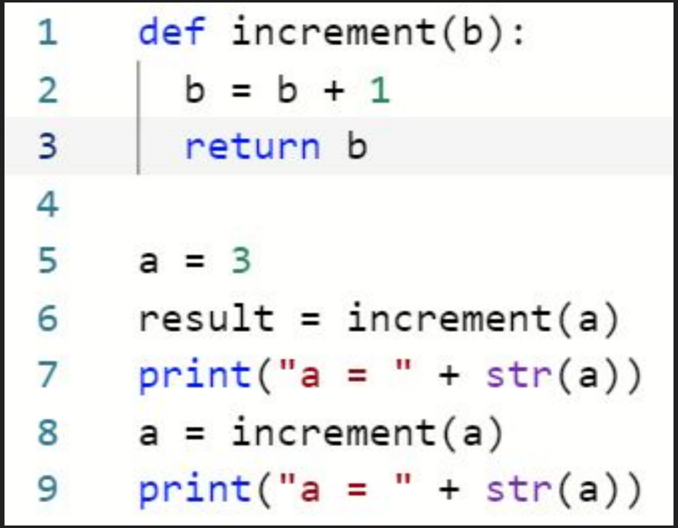


```
a = 3
a = 4
```

Despite sharing the same name, changing the ‘a’ in the function did not directly change the ‘a’ outside of the function

Function Parameter Names

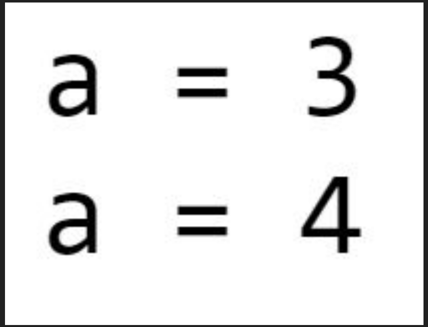
Function parameters names do not need to be the same as the names used in the function call.



```python
def increment(b):
    b = b + 1
    return b

a = 3
result = increment(a)
print("a = " + str(a))
a = increment(a)
print("a = " + str(a))
```

Output:



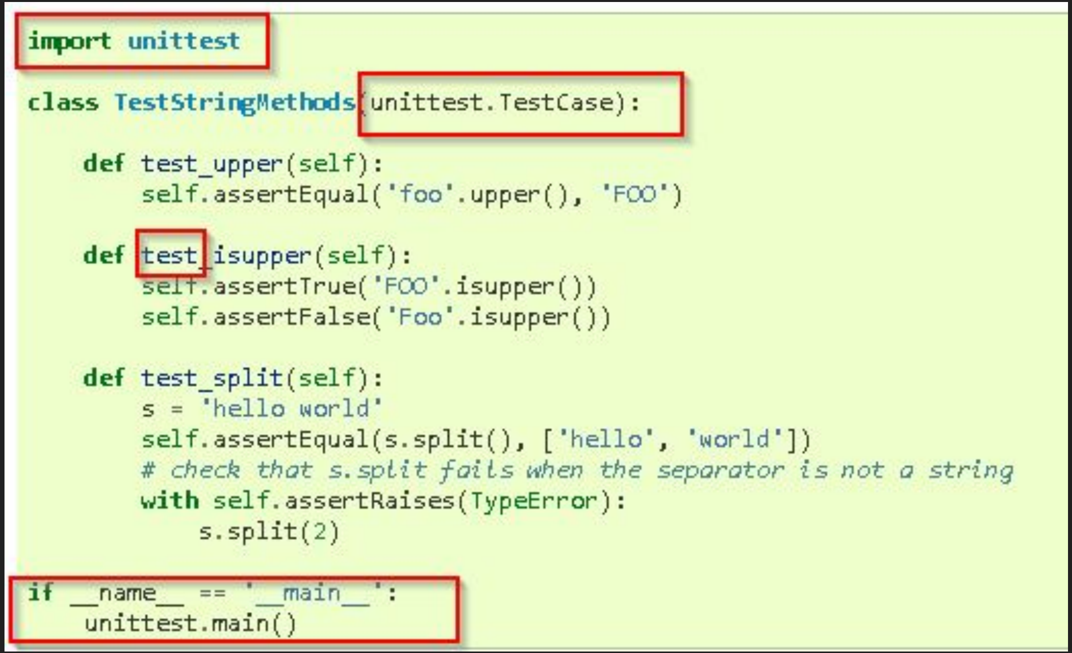
```
a = 3
a = 4
```

Output remained the same

Unit Testing

Contract for expected behavior

- Return values
- Error Raising



```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

Edge Cases

Problem or scenario that is caused as a result of an extreme value.

- Division by 0
- Indexing into an empty string
- ...

Diff Testing

- Compares the contents of two files
- Only way to automate testing I/O
  - Input
  - Print statements

Console

```
~/examples : diff file1 file2
1c1
< Hello
---
> Goodbye
~/examples : diff file1 file3
~/examples :
```

file1

```
Hello
```

file2

```
Goodbye
```

file3

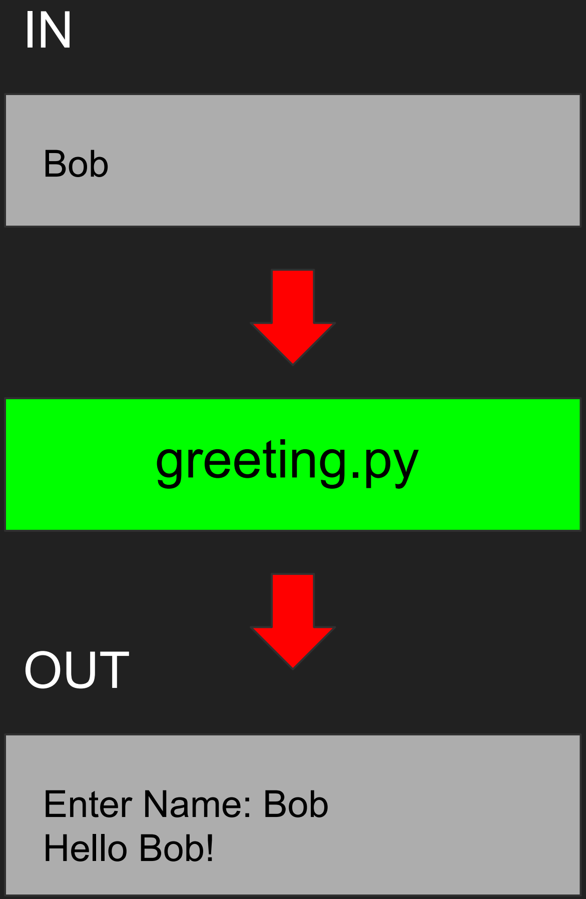
```
Hello
```

Redirection

- Use text from a file as inputs to a program
- Write information to a file instead of the console

Console

```
~/examples : python3 greeting.py
Enter Name: Bob
Hello Bob!
~/examples : python3 greeting.py < IN > OUT
~/examples :
```



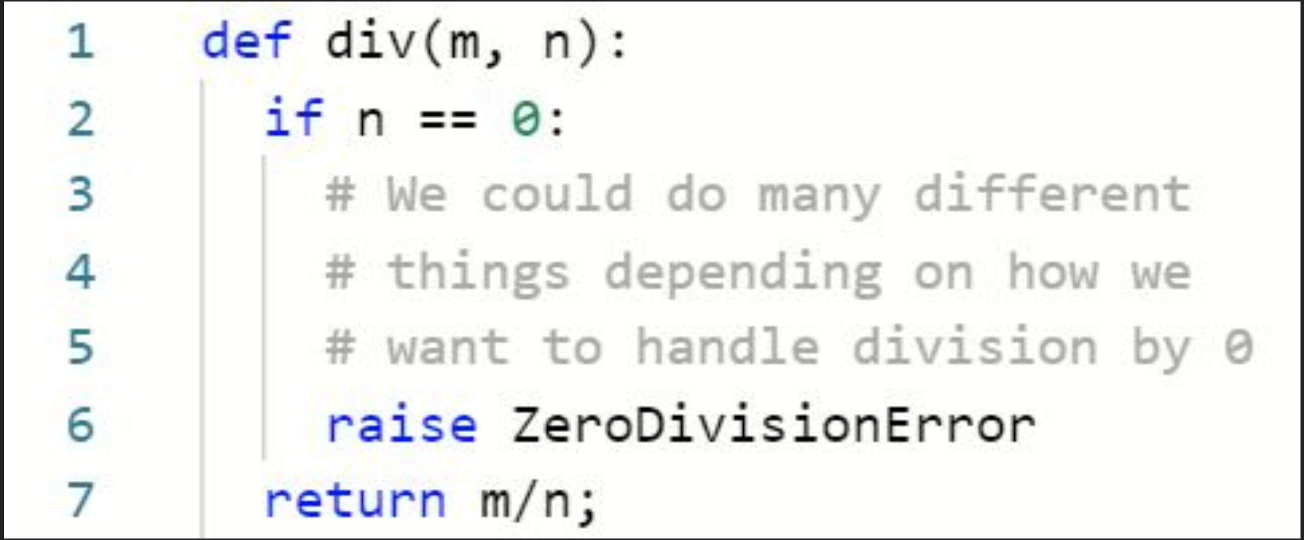
```
Bob
```

```
Enter Name: Bob
Hello Bob!
```

Live Coding Example #1

Create a function that called “div” that divides two given numbers and return the result. Write 3 test cases to validate your solution.

Example Solution



```python
def div(m, n):
    if n == 0:
        # We could do many different
        # things depending on how we
        # want to handle division by 0
        raise ZeroDivisionError
    return m/n;
```

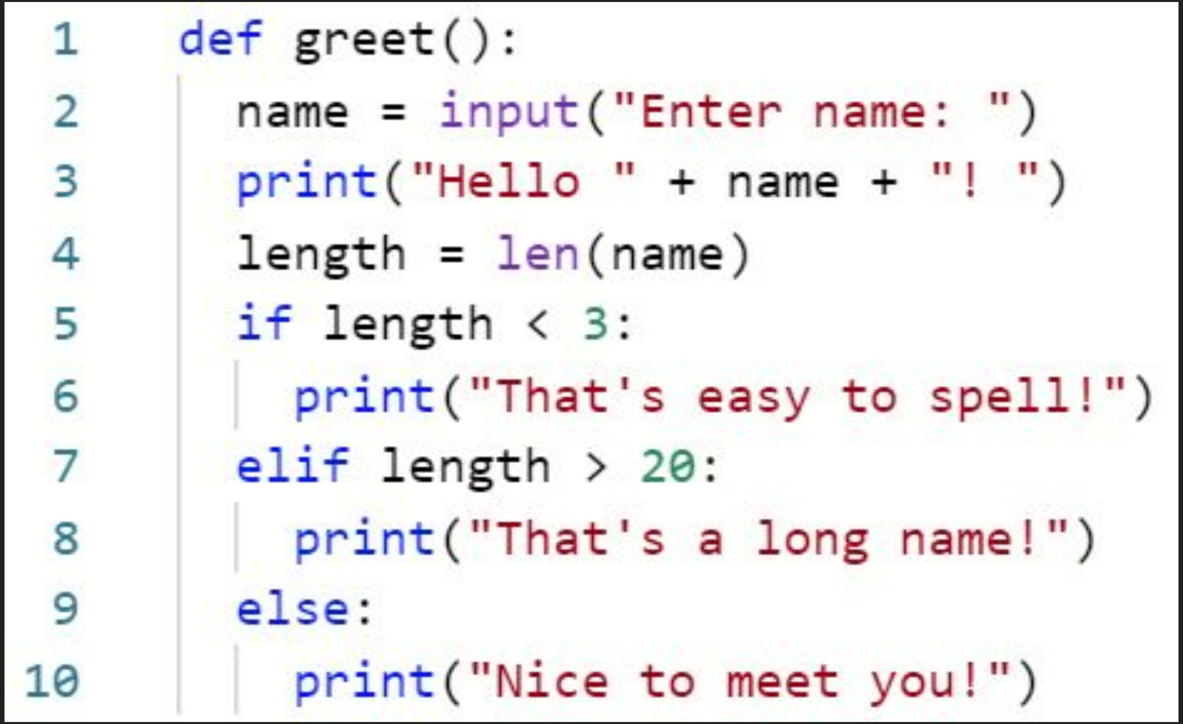
Live Coding Example #2

Write a function called *greet* that asks the user to input their name and then prints “Hello {Name}!” to the console along with a message.

- Names that are < 3 letters print “That’s easy to spell!”
- Names that are > 20 letters print “That’s a long name!”
- All other names print “Nice to meet you!”

Diff test the results for all scenarios.

Example Solution



```python
def greet():
    name = input("Enter name: ")
    print("Hello " + name + "! ")
    length = len(name)
    if length < 3:
        print("That's easy to spell!")
    elif length > 20:
        print("That's a long name!")
    else:
        print("Nice to meet you!")
```

Group Assignment

Class GPA Calculator