Functions

SHARP SIGHT

What you'll learn

- What are functions: reusable code

- Syntax to define functions

- How to modify functions to take inputs (i.e., function parameters)
  - single parameters
  - multiple parameters
  - establishing default arguments (AKA, default parameter values)

- Function outputs

Function basics

Functions are reusable blocks of code

- Functions are like many lines of code collected together
  - we can give a name to these reusable code blocks
- Functions enable you to write code once and reuse it
- Functions can accept inputs
- Functions can also return outputs

Syntax: how to define a new function

The `def` keyword indicates that we are defining a new function

```
def function_name(parameters):
    reusable block of code
```

We must name functions

We must provide a name for the function

```
def function_name(parameters):
    reusable block of code
```

Note that the rules for function names are essentially the same as the rules for variable names

Function naming, continued

Following the name, we must have a set of parenthesis

```
def function_name(parameters):
    reusable block of code
```

Functions can take inputs, which we call parameters

Inside of the parenthesis, we can provide *inputs* to the function, which we call *parameters*

```
def function_name(parameters):
    reusable block of code
```

Parameters are *optional*

We will talk more about parameters in a separate section ...

A function contains a reusable block of code

```
def function_name(parameters):
    reusable block of code
```

Underneath the function definition, you should have a code block

This could be 1 line or many lines!

A function contains a reusable block of code

```
def function_name(parameters):
    reusable block of code
```

This indentation must be present ....

Remember: white space is syntactically meaningful in Python

The best practice is to <u>use 4 spaces to indent code blocks</u>

Function example

Example of a simple function

- This is a very simple example
  - This function does one thing
  - This function prints the string `"Study data science."`

```
def print_advice():
    print("Study data science.")

print_advice()
Study data science.
```

Example of a simple function

The `def` keyword indicates that we are defining function

Here, we're naming the function `print_advice()`

```
def print_advice():
    print("Study data science.")

print_advice()
Study data science.
```

Example of a simple function

```
def print_advice():
    print("Study data science.")

print_advice()
Study data science.
```

Here, we're defining exactly what the function will do

This is a simple example, so it only prints out a sentence ...

Remember though, the code block in a function can do almost anything you want

Example of a simple function

Here, we're using the function (i.e. "calling" the function)

```
def print_advice():
    print("Study data science.")

print_advice()
Study data science.
```

When we call the function, it executes all of the code in the code block

Quick recap of this function example

- We define the function with `def`
  - and give it a name
- Write the code to execute under the function definition
  - Indent the code block!
- You can call the function by name
  - it will execute the code block

Functions that take an input

Function inputs: parameters

- Parameters allow the function to accept inputs
  - parameters are like variables that are used exclusively inside the function code

- Parameters enable you to write code that changes when you call it

Syntax: function parameters

Function parameters are defined inside of the parenthesis, after the function name

```
def function_name(parameters):
    reusable block of code
```

Remember: Parameters are *optional*

Example: function with parameter

Here, we're defining the parameter `target`

```
def hello_target(target):
    print("Hello",target)

hello_target("Cleveland!")
Hello Cleveland!
```

Remember ... parameters are like *variables*

... we can name them whatever we want.

Example: function with parameter

```
def hello_target(target):
    print("Hello",target)

hello_target("Cleveland!")
Hello Cleveland!
```

Here, we're calling the function with a value, `"Cleveland!"`

Calling the function like this is like setting `target = "Cleveland!"`

Example: function with parameter

```
def hello_target(target):
    print("Hello",target)

hello_target("Cleveland!")
Hello Cleveland!
```

The code effectively executes the print statement `print("Hello", "Cleveland!")`

Recap: function parameters

- Parameters let us provide *input* values when we call the function

- Function parameters enable us to subtly change the behavior of the function
  - The parameters act like variables inside the function code

Functions with default parameter values

You can define default values for function parameters

- It's possible to define default values for function parameters

- If you don't specify a value when you call the function, the parameter uses the default

Syntax: function parameters with default values

Here, we're defining the parameter as well as the default value

```
def function_name(parameter = default-value):
    reusable block of code
```

Example: a function parameter with a default value

Here, `subject` is an input parameter.

```
def print_advice(subject = 'data science'):
    print("Study", subject)

print_advice()
Study data science
```

Example: a function parameter with a default value

The string `'data science'` is the default value of the `subject` parameter

```
def print_advice(subject = 'data science'):
    print("Study", subject)

print_advice()
Study data science
```

Example: a function parameter with a default value

If we call the function without a new value, it will execute the code with the default value (the default "argument")

```
def print_advice(subject = 'data science'):
    print("Study", subject)

print_advice()
Study data science
```

Example: a function parameter with a default value

But we can also call the function with a *new* value

```
def print_advice(subject = 'data science'):
    print("Study", subject)

print_advice(subject = 'statistics')
Study statistics
```

When we call a function with a new value for a parameter, that value is passed to the function code when the function is executed

Functions with multiple parameters

Functions with multiple parameters

- Functions with multiple parameters are defined like functions with one parameter
  - specify the parameters inside the parenthesis
  - just separate parameters with commas

- Functions can have a maximum of 256 parameters

Example: a function with multiple parameters

Here, we're defining two parameters, and both of these are passed into the code of the function

```
def hello_both(target1, target2):
    print("Hello", target1, "and", target2)

hello_both("John", "Mike")
Hello John and Mike
```

Functions that return output values

Return statements *return* a value

- You can use the return statement inside of a function to return a value
- The value that's returned is like *the output* of the function
- Return statements are optional
  - Some functions return values, some do not

Syntax: function return values

Here, we're defining a function, which has a code block underneath

```
def function_name():
    reusable block of code
    return …
```

At the end of the code block, we have a *return* statement

The return statement will return a value

Example: a function with a return statement

Here, we're defining a new function the takes an input parameter, `number`

```
def calculate_cube(number):
    return number**3

calculate_cube(3)
27
```

On the second line, we're calculating the cube of the input `number`

Example: a function with a return statement

The `return` statement will return the value of `number**3` (the cube of number) as the output of the function

```
def calculate_cube(number):
    return number**3

calculate_cube(3)
27
```

Example: a function with a return statement

So when we call the function, it will return the value of `number**3`

```
def calculate_cube(number):
    return number**3

calculate_cube(3)
27
```

More notes on return statements

- You can use multiple return statements in a function
  - for example, return different values for different conditions

- Return statements are important for building good functions

- But, return statements are more of an intermediate topic
  - don't worry about them too much in the beginning

Recap

Recap of what we learned

- What are functions: reusable code

- How to define functions

- Defining function parameters
  - multiple parameters
  - default arguments (AKA, default parameter values)

- Function return values