Functions

Lecture 6

Jonathan Kapleau
Function calls

- A function is a named sequence of statements that performs a computation.

- We define a function we specify its name, arguments, and the sequence of statements.

- Later we can call the function by name and the statements will be executed.
Function calls

• We have already seen several functions:
  >>> type(32)
  >>> int('57')
  >>> len('hello')

• It is common to say that a function takes arguments and returns a value
Type conversion functions

- **int()**
  
  ```
  >>> int('1')
  >>> int(3.2)
  ```

- **float()**
  
  ```
  >>> float('3.2')
  >>> float(6)
  ```

- **str()**
  
  ```
  >>> str(1.25)
  ```
Math functions

- Python has a module to perform common mathematical functions
- But before we can use it, we must “import” it
  ```python
  >>> import math
  >>> import math
  ```
- To use a math function you must specify the name of the module and the function
  ```python
  >>> value = math.cos(math.pi)
  >>> value = math.cos(math.pi)
  ```
- To see all of the functions in the math module type 'dir(math)', for help type 'help(math)'
Adding new functions

- A function definition specifies the name of a function and the sequence of statements it performs:
  ```python
def print_hello():
    print('Hello, good morning to you!')
```
- 'def' is a keyword that specifies that this is a function definition.
- The empty parentheses after the name tell us the function doesn't take any arguments.
Adding new functions

- The first line is called the 'header' and the rest is the 'body'
  - The header must end with a colon
  - The body must be indented
- Now we can call, or invoke, the function:
  >>> print_hello()
  Hello, good morning to you!
- We call our new function the same way we call built-in functions
Flow of execution

- A function must be defined before it can be used.

- Execution begins with the first statement and are executed one at a time, in order, from top to bottom.

- Function definitions are not executed until the function is invoked.
Parameters and arguments

• Some functions require arguments
• Inside the function, the arguments are assigned to variables called parameters:

```python
def print_message_twice(message):
    print(message)
    print(message)
```

• The function assigns the argument to the parameter named message
Parameters and arguments

- When the function is called, the 'message' parameter, whatever it is, will be printed twice:

```python
>>> greeting = 'Hello, good morning!'
>>> print_message_twice(greeting)
Hello, good morning!
Hello, good morning!
```
Variables and parameters are local

- When we create a variable inside of a function, it is local, which means it only exists inside of the function.

- Parameters are also local to the function they are defined for.
Value-returning functions

- Some of the functions return a value

- When you invoke a value-returning function you should assign the result to a variable for later use

  >>> x = math.cos(0.0)
  >>> print(x)
  1.0
Void functions

- Functions that don't return a value are called void functions.

- They generally “do something”

- If you assign the result to a variable you get the special value 'None'.
Return Values

- Some functions produce results
- Calling the function generates a value that can be assigned to a variable
- We use the 'return' statement to return a value to the calling statement
Return Statements

- In a value-returning function the 'return' statement includes an expression.

- The return statement tells our program to stop executing the function and return to where it was called.

- The value that is returned can be assigned to a variable.
Return Statements

• For example:

def area(radius):
    return math.pi * radius ** 2

• We can call the function like this:
  circleArea = area(3.0)

• After, the variable holds the value
Return and If Statements

- Return statements can be put in a conditional:

```python
def absolute_value(x):
    if x < 0:
        return -x
    else:
        return x
```

- Only one return will be executed
Boolean Functions

- A function can return the value 'True' or 'False'

- In this way we can hide complicated tests and simplify our programs
Why functions?

- Makes programs easier to read and debug
- Eliminates repetition in our code
- Re-usability