[301] Creating Functions

Tyler Caraza-Harter

Learning Objectives Today

Learn how to create functions:

- Map algebraic notation to Python
- Take multiple parameters
- Set default arguments
- Differentiate between output to screen and output via return values
- Understand indentation

Please continue reading Chapter 3 of Think Python

Also read “Creating Fruitful Functions”

Modules:

- How to save your functions in modules

Flow of execution:

- Trace through execution
- Understand functions that call other functions
- Differentiate definition time vs invocation time

Main Code:

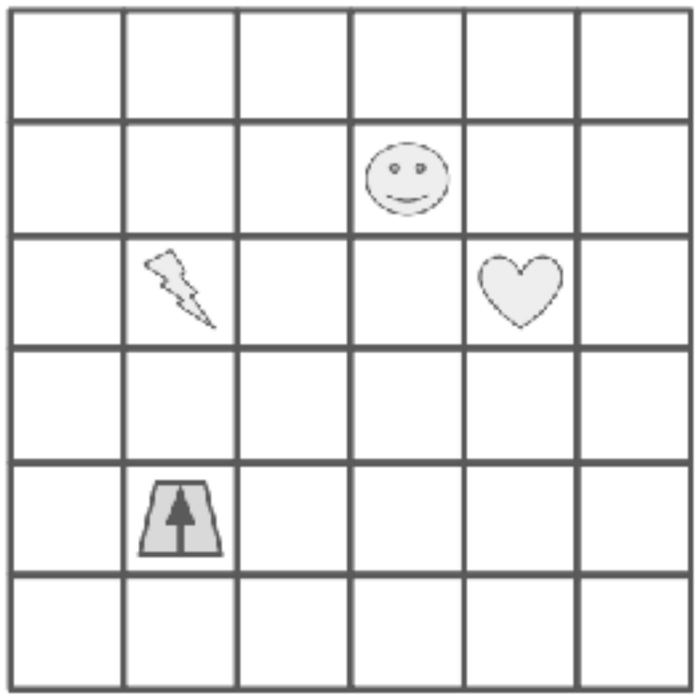
1. Put 2 in the “moves” box
2. Perform the steps under “Move Code”, then continue to step 3
3. Rotate the robot 90 degrees to the right (so arrow points to right)
4. Put 3 in the “moves” box
5. Perform the steps under “Move Code”, then continue to step 6
6. Whatever symbol the robot is sitting on, write that symbol in the “resut” box

Move Code:

A. If “moves” is 0, stop performing these steps in “Move Code”, and go back to where you last were in “Main Code” to complete more steps
B. Move the robot forward one square, in the direction the arrow is pointing
C. Decrease the value in “moves” by one
D. Go back to step A

how do we write functions like move code?

Functions are like “mini programs”, as in our robot worksheet problem



Types of functions

Sometimes functions **do** things

- Like “Move Code”
- May produce output with print
- May change variables

Sometimes functions **produce** values

- Similar to mathematical functions
- Many might say a function “returns a value”
- Downey calls these functions “fruitful” functions
  (we’ll use this, but don’t expect people to generally be aware of this terminology)

Sometimes functions do both!

Types of functions

Sometimes functions **do** things

- Like “Move Code”
- May produce output with print
- May change variables

Sometimes functions **produce** values

- Similar to mathematical functions
- Many might say a function “returns a value”
- Downey calls these functions “fruitful” functions
  (we’ll use this, but don’t expect people to generally be aware of this terminology)

Sometimes functions do both!

Math to Python

Math: $f(x) = x^2$

Math:

```
def f(x):
    return x ** 2
```

Math to Python

Math: $\boxed{f}(x) = x^2$

Math:

```
def f(x):
    return x ** 2
```

Function name is “f”

Math to Python

Math: $f(\boxed{x}) = x^2$

Math:

```
def f(x):
    return x ** 2
```

It takes one parameter, "x"

Math to Python

Math: $\boxed{f(x) =} x^2$

Math:

```
def f(x):
    return x ** 2
```

In Python, start a function definition with "def" (short for definition), and use a colon (":") instead of an equal sign ("=")

Math to Python

Math: $f(x) = \boxed{x^2}$

Math:

```
def f(x):
    return x ** 2
```

In Python, put the "return" keyword before the expression associated with the function

Math to Python

Math: $f(x) = x^2$

Math:

```
def f(x):
    return x ** 2
```

In Python, indent before the expression (or statements)

Math to Python

Math: $g(r) = \pi r^2$

Math:

```
def g(r):
    return 3.14 * r ** 2
```

Computing the area from the radius

Math to Python

Math: $g(r) = \pi r^2$

Math:

```
def get_area(radius):
   return 3.14 * radius ** 2
```

In Python, it's common to have longer names for functions and arguments

Math to Python

Math: $g(r) = \pi r^2$

Math:
```
def get_area(diameter):
    radius = diameter / 2
    return 3.14 * radius ** 2
```

It's also common to have more than one line of code (all indented)

demos for rest of lecture