Think Python 2e, Chapter 3 Notes

Geoffrey Matthews

September 13, 2022

Function calls

- Functions take an argument and return a result.

```
1 >>> type(42)
2 <class 'int'>
```

- The function is `type`
- The argument is `42`
- the return value is `<class 'int'>`

Functions for changing type

```
32
>>> int('Hello')
ValueError: invalid literal for int(): Hello
>>> int(3.99999)
3
>>> int(-2.3)
-2
>>> str(32)
'32'
>>> str(3.14159)
'3.14159'
```

Math functions

- Most math functions are in the math **module**.
- You can only use them if you import the module:

```
>>> import math
>>> dir(math)['__doc__', '__loader__', '__name__',
    '__package__', '__spec__', 'acos', 'acosh', '
   asin', 'asinh', 'atan', 'atan2', 'atanh', '
   ceil', 'comb', 'copysign', 'cos', 'cosh', '
   degrees', 'dist', 'e', 'erf', 'erfc', 'exp', '
   expm1', 'fabs', 'factorial', 'floor', 'fmod',
   'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf
   ', 'isclose', 'isfinite', 'isinf', 'isnan', '
   isqrt', 'lcm', 'ldexp', 'lgamma', 'log', '
   log10', 'log1p', 'log2', 'modf', 'nan', '
   nextafter', 'perm', 'pi', 'pow', 'prod', '
   radians', 'remainder', 'sin', 'sinh', 'sqrt',
   'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

Help on buil-in functions

```
>>> help(math.sin)
Help on built-in function sin in module math:

sin(x, /)
    Return the sine of x (measured in radians).
```

Beware of floating point numbers!

- What do you think this returns?

```
1 >>> math.sin(math.pi)
```

Beware of floating point numbers!

- What do you think this returns?

```
>>> math.sin(math.pi)
```

```
1.2246467991473532e-16
```

Beware of floating point numbers!

- What do you think this returns?

```
>>> math.sin(math.pi)
```

```
1.2246467991473532e-16
```

- How about this?

```
>>> math.exp(math.log(22))
```

Beware of floating point numbers!

- What do you think this returns?

```
>>> math.sin(math.pi)
```

```
1.2246467991473532e-16
```

- How about this?

```
>>> math.exp(math.log(22))
```

```
22.000000000000004
```

Beware of floating point numbers!

- What do you think this returns?

```
>>> math.sin(math.pi)
```

```
1.2246467991473532e-16
```

- How about this?

```
>>> math.exp(math.log(22))
```

```
22.000000000000004
```

- How about this?

```
>>> math.cos(math.pi)
```

Beware of floating point numbers!

- What do you think this returns?

```
>>> math.sin(math.pi)
```

```
1.2246467991473532e-16
```

- How about this?

```
>>> math.exp(math.log(22))
```

```
22.000000000000004
```

- How about this?

```
>>> math.cos(math.pi)
```

```
-1.0
```

Floating point numbers are almost never exact!

Defining new functions

```
1 def print_lyrics():
2     print("I'm a lumberjack, and I'm okay.")
3     print("I sleep all night and I work all day.")
```

- The name of the function is `print_lyrics`
- The rules for variable and function names are the same.
- The empty parentheses indicate that this function takes no arguments.
- The first line is called the **header**
- The rest is called the **body**
- Double quotes let us use single quotes as apostrophes.
- Normally entered in script mode.

Functions can call other functions

- Create a module (write a file) with the following two function definitions.

```
1 def print_lyrics():
2     print("I'm a lumberjack, and I'm okay.")
3     print("I sleep all night and I work all day.")
4 def repeat_lyrics():
5     print_lyrics()
6     print_lyrics()
```

- Now run the module.
- Nothing happens, because you haven't printed anything.
- Defining a function does not run any code.
- You have to call the functions to print anything.

Calling the functions

- In the shell, you can call the functions:

```
>>> print_lyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.

>>> repeat_lyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

Defining and calling in the script

- You define the following module, and run it. What happens?

```
1 def print_lyrics():
2     print("I'm a lumberjack, and I'm okay.")
3     print("I sleep all night and I work all day.")
4 def repeat_lyrics():
5     print_lyrics()
6     print_lyrics()
7
8 repeat_lyrics()
```

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

__main__

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

__main__

__main__ →f

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

__main__

__main__ →f

__main__ →f →g

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

__main__

__main__ →f

__main__ →f →g

__main__ →f →g →h

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

```
__main__
__main__ →f
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
```

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

__main__

__main__ →f

__main__ →f →g

__main__ →f →g →h

__main__ →f →g →h →i

__main__ →f →g →h

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

```
__main__
__main__ →f
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
```

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

```
__main__
__main__ →f
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f →g →i
```

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

```
__main__
__main__ →f
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f →g →i
__main__ →f →g
```

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

```
__main__
__main__ →f
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f →g →i
__main__ →f →g
__main__ →f →g →h
```

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

```
__main__
__main__ →f
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f →g →i
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
```

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

```
__main__
__main__ →f
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f →g →i
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
```

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

__main__
__main__ →f
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f →g →i
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

```
__main__
__main__ →f
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f →g →i
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f
```

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

```
__main__
__main__ →f
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f →g →i
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f
__main__ →f →i
```

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

```
__main__
__main__ →f
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f →g →i
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f
__main__ →f →i
__main__ →f
```

Flow of control and stack frames

```
1  def f():
2      print(1)
3      g()
4      i()
5      print(2)
6  def g():
7      print(3)
8      h()
9      i()
10     h()
11     print(4)
12 def h():
13     print(5)
14     i()
15     print(6)
16 def i():
17     print(7)
18 f()
```

```
__main__
__main__ →f
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f →g →i
__main__ →f →g
__main__ →f →g →h
__main__ →f →g →h →i
__main__ →f →g →h
__main__ →f →g
__main__ →f
__main__ →f →i
__main__ →f
__main__
```

Parameters and arguments

```
def print_twice(bruce):
    print(bruce)
    print(bruce)
```

```
>>> print_twice('Spam')
Spam
Spam
>>> print_twice(42)
42
42
>>> print_twice(math.pi)
3.14159265359
3.14159265359
>>> print_twice('Spam '*4)
Spam Spam Spam Spam
Spam Spam Spam Spam
>>> print_twice(math.cos(math.pi))
-1.0
-1.0
```

Functions can be passed to functions

```
1 def foo():
2     print('foo')
3
4 def do_twice(f):
5     f()
6     f()
7
8 do_twice(foo)
```

Functions can be passed to functions

```
1 def print_fancy(x):
2     print('==> ' + x + ' <==')
3
4 def do_twice(f, x):
5     f(x)
6     f(x)
7
8 do_twice(print_fancy, 'Geoffrey the Magnificent')
```

Parameters and variables in a function are local

```
1 x = 11
2
3 def foo(y):
4     z = 2*y
5     print(x,y,z)
6
7 foo(2*x)
8 print(x,y,z)
```

Parameters and variables in a function are local

```
1 x = 11
2
3 def foo(y):
4     z = 2*y
5     print(x,y,z)
6
7 foo(2*x)
8 print(x,y,z)
```

__main__	x	→	11
foo	y	→	22
	z	→	33

Stack frames

- If you're in the box you can see it!
- If you're not in the box you can't see it!

Parameters and variables in a function are local

```
1  x = 11
2  def foo(y):
3      z = 2*y
4      print(x,y,z)
5      bar(2*z)
6  def bar(x):
7      y = 2*x
8      print(x,y,z)
9  foo(x)
10 print(x,y,z)
```

Parameters and variables in a function are local

```
1  x = 11
2  def foo(y):
3      z = 2*y
4      print(x,y,z)
5      bar(2*z)
6  def bar(x):
7      y = 2*x
8      print(x,y,z)
9  foo(x)
10 print(x,y,z)
```

__main__	x	→	11
foo	y	→	22
	z	→	33
bar	x	→	66
	y	→	132

Stack frames

- If you're in the box you can see it!
- If you're not in the box you can't see it!

Fruitful functions

```
>>> math.sqrt(5)
2.236067977499789
```

If you're in a script, the value is lost forever:

```
math.sqrt(5)
```

You've got to do something with it:

```
print(math.sqrt(5))
x = math.sqrt(5)
```

Functions that return nothing return `None`

```
>>> x = print('hello')
hello
>>> print(x)
None
```

Defining your own fruitful functions

```
1 def smallest_digit(n):
2     return n % 10
3
4 def average(x,y):
5     return (x+y)/2
```

Functions are your friends

- Naming a chunk of code makes your program easier to read.
- Eliminate redundancy.
- You can debug one function at a time.
- Errors give you a stack trace.
- Functions can be used in many different programs.

Vocabulary

function: A named sequence of statements that performs some useful operation. Functions may or may not take arguments and may or may not produce a result.

function definition: A statement that creates a new function, specifying its name, parameters, and the statements it contains.

function object: A value created by a function definition. The name of the function is a variable that refers to a function object.

header: The first line of a function definition.

body: The sequence of statements inside a function definition.

Vocabulary

parameter: A name used inside a function to refer to the value passed as an argument.

function call: A statement that runs a function. It consists of the function name followed by an argument list in parentheses.

argument: A value provided to a function when the function is called. This value is assigned to the corresponding parameter in the function.

local variable: A variable defined inside a function. A local variable can only be used inside its function.

return value: The result of a function. If a function call is used as an expression, the return value is the value of the expression.

Vocabulary

fruitful function: A function that returns a value.

void function: A function that always returns None.

None: A special value returned by void functions.

module: A file that contains a collection of related functions and other definitions.

import statement: A statement that reads a module file and creates a module object.

module object: A value created by an import statement that provides access to the values defined in a module.

dot notation: The syntax for calling a function in another module by specifying the module name followed by a dot (period) and the function name.

Vocabulary

composition: Using an expression as part of a larger expression, or a statement as part of a larger statement.

flow of execution: The order statements run in.

stack diagram: A graphical representation of a stack of functions, their variables, and the values they refer to.

frame: A box in a stack diagram that represents a function call. It contains the local variables and parameters of the function.

traceback: A list of the functions that are executing, printed when an exception occurs.