CSC110 LEC9201 Lecture 3 Notes

Anatoly Zavyalov

September 16, 2020

1 Mathematical Definition of a Function

Recall a mathematical definition of a function: $f : A \to B$, where A is the domain, and B is the codomain.

For example: $f : \mathbb{R} \to \mathbb{R}, f(x) = x^2$.

Functions take in inputs and return outputs: $f(5) = 25$.

2 Functions in Python

In Python, functions do the same thing: take in inputs and return a value. But they aren't just limited to numbers!

2.1 Built-in Python Functions

Some of Python's built-in functions:

- `abs`: returns the absolute value of an `int` / `float`
  For example: `abs(-1)` returns `-1`.
- `len`: returns the number of elements in a collection (this can be a set, string, tuple, list, etc.)
  For example: `len(1, 2, 3, 1000)` returns `4`.
- `sum`: returns the sum of elements in a collection
  For example: `sum([1, 2, 3, 100])` returns `106`.
- `sorted`: returns a list of sorted elements of a collection
  For example: `sorted([3, 1, 100, -5])` returns `[-5, 1, 3, 100]`.
- `max`: returns the maximum number in a collection
  For example: `max([3, 1, 100, 69])` returns `100`, and `max({1: 100, 2: -200})` returns `2`.
- `type`: returns the 'type' of a literal/variable
  For example: `type(1)` returns `<class 'int'>`, and `type([1, 2, 3])` returns `<class 'list'>`.

2.2 Terminology

Consider this code:

```
>>> abs(-5)
5
```

- `abs(-5)` is a **function call expression**
- `abs` is the **function** being called
- `-5` is an **argument**; it is **passed** to `abs`
- `abs` **returns** `5`; `abs(-5)` **evaluates to** `5`

2.3 A closer look at `max`

```
>>> max([1, 2, 3])
3
```

```
>>> max(100, 200)
200
>>> max(1, 2, 3, 5, 34548937534753485, 2, 3, 5, 7,
   4)
34548937534753485
```

Note that when calling `max`, you can have more than one argument, functioning the same way as if you passed a list.

3 Methods: Data type-specific functions

A **method** is a function that has been defined within a data type. Note that every method is a function, but not the other way around.

For example:

- `str.lower` converts all characters in a **string** to lowercase.

```
>>> str.lower('Anatoly')
'anatoly'
```

- `str.split` returns a list containing each 'word' of a **string**, split by space characters.

```
>>> str.split('Anatoly is very cool please
    believe me ;(')
['Anatoly', 'is', 'very', 'cool', 'please', '
    believe', 'me', ';(']
```

- `set.union` returns a union of a `set`.

```
>>> set.union({1, 2, 3}, {2, 4, 9})
{1, 2, 3, 4, 9}
```

- `list.count` takes a **list** as an argument, as well as another value, and checks how many times the other value is within the list.

```
>>> list.count([1, 2, 3, 3, 3, 3, 3, 3, 5], 3)
6
```

4 Defining our own functions in Python

Recall $f : \mathbb{R} \to \mathbb{R}, f(x) = x^2$. How do we write this in Python?

Here's how:

```
1 def square(x: float) -> float:
2     """Return x squared.
3
4     >>> square(3.0)
5     9.0
6     >>> square(2.5)
7     6.25
8     """
9     return x ** 2
```

Let's analyze this code:

Line 1: **The function header**

- `def` is the syntax in Python for starting a function definition
- `square` is the name of the function
- `x` is the name of the parameter
- `float` (the one in brackets) is the type of the parameter (called a **type annotation**). The Python interpreter is not checking for type annotations, however it does improve readability.
- `float` (the one outside the brackets) represents the return type of the function. Type annotations are required in this course.

Lines 2-8: **Function docstring**/comment

- This description of the function, anything within the docstring is **not** executed by the Python Interpreter.
- Writing docstrings is highly recommended to develop good practices.
- When calling `help(square)`, the function docstring is returned (which is super helpful!).

Line 9: **The function body**

- This is the actual code that is executed when the function is run.
- The `return` statement returns the value of `x ** 2`.

5 Local Variables and Function Scope

5.1 Scope

The **scope** of a variable is where in the code it can be accessed.

Functions have **local scope**: their variables can only be accessed inside the function body. Note that all function parameters have local scope, and can only be accessed within the function body.

Variables that have local scope are called **local variables**.

Consider the following code:

```
1  def square(x: float) -> float:
2      """Return x squared.
3
4      >>> square(3.0)
5      9.0
6      >>> square(2.5)
7      6.25
8      """
9      return x ** 2
10
11 >>> n = 10.0
12 >>> result = square(n + 3.5)
```

For this code, the following is the memory model:

Variable	Value
`n`	`10`

Table 1: Local variables in `__main__` (console)

Variable	Value
`x`	`13.5`

Table 2: Local variables in `square`