Functions

Table of contents

See csse1001[1] for course logistics — this note covers Lecture 3A's technical content.

Today's outline

- What a function is and how to define one
- The `return` statement, and how it differs from `print`
- Improving functions with type hints and docstrings

Learning objectives

1. User-defined functions bundle code together and are the building blocks of more sophisticated programs.
2. `return` hands back a value and exits a function; it is not the same as `print`.
3. Functions can be documented with type hints and docstrings.

[1] courses/csse1001/csse1001.pdf

What is a function?

We have already used functions — `*` and `max`, for example — each of which takes input and returns output. User-defined functions let us name and reuse our own bundles of code, just like a mathematical function such as $f(x) = x^2 + x + 1$:

```
>>> def f(x):
...     return x**2 + x + 1
>>> f(3)
13
```

See python-functions[2] for the full syntax rules (indentation, multiple parameters, the `return` statement, `return` vs. `print`, type hints, and docstrings) — used throughout the rest of this note.

Building up Heron's formula

Exercise. The area of a triangle with side lengths a, b, c is $\sqrt{s(s-a)(s-b)(s-c)}$ where $s = \frac{1}{2}(a+b+c)$ (Heron's formula). What is the area of the triangle with sides 3, 4, and 5?

As a calculator, we'd have to type this out in full, and it would be tedious to repeat for other side lengths:

```
>>> (0.5*(3+4+5)
...  *(0.5*(3+4+5)-3)
...  *(0.5*(3+4+5)-4)
...  *(0.5*(3+4+5)-5))**0.5
6.0
```

Using names and sequencing (see 2025-08-04-python-memory-model[3]) avoids repeating the same sub-expression:

```
>>> a, b, c = 3, 4, 5
>>> s = (a + b + c)/2
>>> (s*(s-a)*(s-b)*(s-c))**0.5
6.0
```

Wrapping it in a function makes it reusable for *any* triangle:

[2] courses/csse1001/python-functions.pdf
[3] courses/csse1001/2025-08-04-python-memory-model.pdf

```
>>> def heron(a, b, c):
...     s = (a + b + c)/2
...     return (s*(s-a)*(s-b)*(s-c))**0.5
>>> heron(3, 4, 5)
6.0
```

Improving the function

Adding type hints and a docstring (see python-functions[4]):

```
>>> def triangle_area(a: float, b: float, c: float) -> float:
...     """
...     Return the area of the triangle with sides length <a>,
...     <b>, and <c>.
...     Preconditions: <a>, <b>, <c> are all nonzero positive.
...     >>> triangle_area(3, 4, 5)
...     6.0
...     """
...     s = (a+b+c)/2
...     return (s*(s-a)*(s-b)*(s-c))**0.5
>>> triangle_area(2.2, 3.3, 4.4)
3.5147323866832307
```

See python-pep8-style-guide[5] for the naming, spacing, and line-length conventions expected of functions like this.

`return` vs. `print`: quick check

```
>>> def example(x):
...     print(1*x)
...     return 3*x
...     print(2*x)   # never runs -- return already exited the function
>>> a = example(1)
1
>>> a
3
```

[4] courses/csse1001/python-functions.pdf
[5] courses/csse1001/python-pep8-style-guide.pdf

```
>>> def foo(x):
...     if x > 0:
...         print("Positive")
...     if x > 10**5:
...         print("Large positive")
>>> ans = foo(10**6)
Positive
Large positive
>>> type(ans)
<class 'NoneType'>
```

`foo` above never hits a `return`, so it defaults to returning `None` even though it printed something. Contrast with a version that returns instead:

```
>>> def bar(x):
...     if x > 0:
...         return "Positive"
...     if x > 10**5:
...         return "Large positive"
>>> ans = bar(10**6)
>>> ans
'Positive'
```

`bar` exits at the very first `return` it reaches, so `"Large positive"` is never returned even though `x > 10**5` is also true.

Exercise: the middle number

Write a function that takes three integers and returns the number that is not the largest or the smallest:

```
def middle_number(x: int, y: int, z: int) -> int:
    """
    Return the number that is not the largest or smallest
    among the three inputs.
    Precondition: the numbers are distinct.
    >>> middle_number(3, 1, 2)
    2
    >>> middle_number(2, 3, 1)
    2
    """
    return (x + y + z) - min(x, y, z) - max(x, y, z)
```

Summary

Blocks of code can be grouped into functions. Functions take zero or more inputs and hand back a single value designated by `return`.

Next lecture

2025-08-11-sequence-selection-and-iteration[6] — selection.

[6] courses/csse1001/2025-08-11-sequence-selection-and-iteration.pdf