Functions

Table of contents

Outline 1
Learning objectives 1
What is a function? 2
Building up Heron's formula 2
`return`, and what it does to the lines below it 5
Exercise: the middle number 6
Two worked questions 6
Style: PEP8 7
Summary 7
Next lecture 7

Lecture 3A. See csse1001[1] for course logistics.

Outline

- What a function is and how to define one
- The `return` statement, and how it differs from `print`
- Improving a function with type hints and docstrings
- PEP8: how the course expects that code to be laid out

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 the function**; it is not the same as `print`.
3. Functions can be documented with type hints and docstrings, neither of which Python enforces.

[1] courses/csse1001/csse1001.pdf

What is a function?

User-defined functions are perhaps the single most important feature of programming languages. They let us **abstract away** complexity by **bundling** lines of code together, and are themselves the building blocks of more sophisticated functions.

We have *already* been using functions — `*` and `max`, for example. Each takes **input** and returns **output**: multiplication takes two numbers and returns a number; `max` takes two or more numbers and returns a number.

Python functions are analogous to those of mathematics and can also be named. A parabola in mathematics is given by the function $f(x) = x^2 + x + 1$, and in Python we write

```
>>> 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 — all of which are used throughout the rest of this note.

Building up Heron's formula

One function, built three times over, each version better than the last. The exercise:

> **Exercise (Heron's formula).** The area of $\triangle ABC$ is $\sqrt{s(s-a)(s-b)(s-c)}$ where $s = \frac{1}{2}(a+b+c)$. *Write a function which computes the area of a triangle from the side lengths a, b, c.*

Question. What is the area of the triangle with sides 3, 4, and 5?

Answer 1 — Python as a calculator. Everything inline, `s` spelled out four times over:

```
>>> (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
```

We have to type a *lot*. It would be hard to repeat.

Answer 2 — using names and sequencing. Naming `s` once removes the repetition (see 2025-08-04-python-memory-model[3]):

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

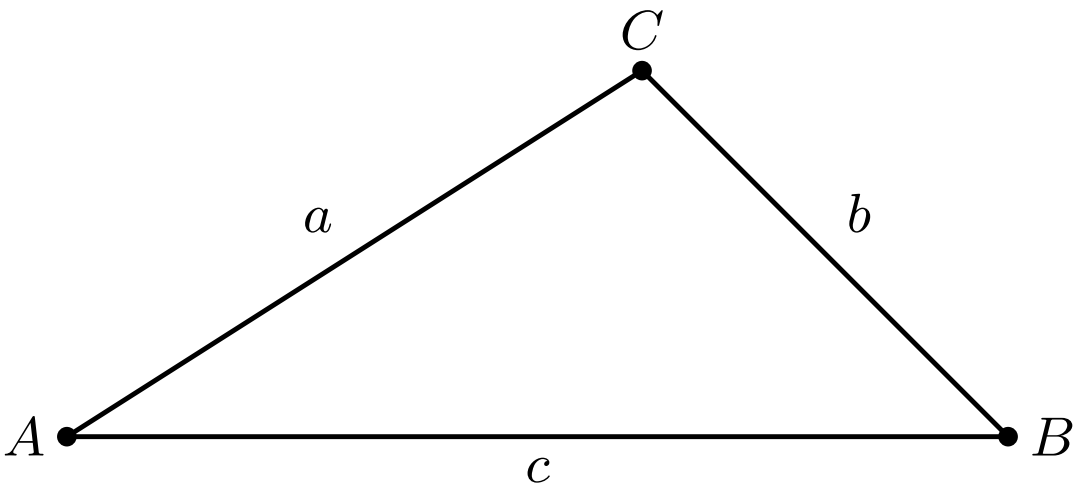

Figure 1: The triangle and its side labels: a on the side from A to C, b on the side from C to B, c on the side from A to B. That is not the usual convention, under which each side is named for the vertex opposite it — here only c is opposite its namesake vertex, while a sits alongside A and b alongside B. Heron's formula is symmetric in a, b and c so this changes no arithmetic, but it changes what you should picture when the code says a.

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

But the names are now stuck at those three values — a different triangle means retyping the lot.

Answer 3 — using functions. The same three lines, parameterised and reusable for *any* triangle:

```
>>> 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
```

The result is `6.0`, not `6`: `**0.5` produces a float.

Improving the function: type hints

In mathematics we would define a mapping

$$\text{triangle_area} : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}.$$

In Python we can write that mapping into the `def` line itself:

```
>>> def triangle_area(a: float, b: float, c: float) -> float:
...     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
```

Type hints are **not enforced**. They are documentation, for making the code more readable — annotate the same function `str` throughout and it still returns the same float. See python-functions[4].

[4] courses/csse1001/python-functions.pdf

Improving the function: docstrings and examples

```
>>> 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
```

That is the full shape the course expects of a Python function: `def` line with type hints, docstring with a short description, preconditions, and a worked example, then the body, then `return`.

`return`, and what it does to the lines below it

`return` is a **reserved word** — one that cannot be assigned by us. It designates what value the function should return *while also terminating the function itself.*

The striking case is the one where the `return` is the *first* line:

```
>>> def example(x):
...     return 3*x
...     return 2*x
...     print(1*x)

>>> a = example(1)
>>> a
3
```

Neither the second `return` nor the `print` ever runs. The other variants of this question, and the diagram of what control actually does, are in python-functions[5].

A function with **no** `return` at all returns `None`:

[5] courses/csse1001/python-functions.pdf

```
>>> def example(x):
...     print(2*x)
>>> a = example(1)
2
>>> a
>>> type(a)
<class 'NoneType'>
```

The default return value is `None`, and `print` is exactly this kind of function: it returns nothing, and its *side effect* is that it displays something.

Exercise: the middle number

> **Exercise.** Write a function that takes three integers and returns the number which 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)
```

The trick is that the three numbers sum to (smallest + middle + largest), so subtracting `min` and `max` leaves the middle one — no comparisons or `if`-statements needed.

Two worked questions

What is the type of `foo`'s return value?

```
def foo(radius: int):
    """
    Precondition:  radius > 0
    """
    area = 3.14159 * radius**2
```

`NoneType`. There is no explicit `return` statement, so the computed area is thrown away.

Is this valid Python code?

```
def foo(x: int, y: int) -> int:
    """
    >>> foo(2, 3)
    6
    """
    return x
ans = foo("oi", 3)
```

Yes. Type hints are *not* enforced. The docstring's own example is wrong too — this `foo` returns `x`, so `foo(2, 3)` is `2`, not `6` — and Python will not complain about that either unless the doctests are actually run. See python-testing[6].

Style: PEP8

The style rules apply to every function above. A **PEP** is a Python Enhancement Proposal — a design document on how to *style* our code — and this course follows Google's PEP, a variant of PEP8. In brief: names start with a letter, are lowercase and underscore-separated; lines stay strictly under 80 characters, wrapped inside a redundant enclosing bracket; and a comment should explain something not obvious in the design rather than restate the line it sits on. The yes/no tables for naming and spacing, and the rule behind the spacing one, are in python-pep8-style-guide[7].

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[8] — selection.

[6] courses/csse1001/python-testing.pdf
[7] courses/csse1001/python-pep8-style-guide.pdf
[8] courses/csse1001/2025-08-11-sequence-selection-and-iteration.pdf