Python Functions

Table of contents

User-defined functions bundle lines of code together so they can be reused, abstracting away complexity. Like mathematical functions, they take input and return output (we've already used built-in functions this way, e.g. `*` and `max`).

Defining a function

A mathematical function such as $f(x) = x^2 + x + 1$ is written in Python as:

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

Functions can take multiple parameters, and calls can be nested inside other expressions:

```
>>> def f(x, y):
...     return x*y
>>> f(-f(5, 2) + 12, f(2, 3))
12
```

Indentation

Four spaces of indentation are significant in Python — they associate a line of code with the control structure above it (here, the function body with its `def`). Inconsistent indentation raises `IndentationError: unexpected indent`.

The `return` statement

`return` is a reserved word (not a function) that hands a value back to the caller and immediately exits the function — any code after the first `return` a call actually reaches (including further `print`s or `return`s) never runs.

If a function has no `return` statement, Python presumes a `return None` as its last line.

`return` versus `print`

`print` displays something as a side effect but *returns* `None`. This looks similar to `return` but behaves very differently once the result is used in further computation:

```
>>> def f(x, y):
...     print(x+y)
>>> a = f(2, 3)
5
>>> b = f(3, 4)
7
>>> a + b
TypeError: unsupported operand type(s) for +: 'NoneType' and 'NoneType'
```

```
>>> def f(x, y):
...     return x + y
>>> a = f(2, 3)
>>> b = f(3, 4)
>>> a + b
12
```

`return` is a reserved word, not a function — we write `return x + y`, not `return(x + y)` (the latter happens to still work, since the brackets are just a redundant grouping, but it's misleading).

Type hints

Type hints annotate the expected type of each parameter and the return value, e.g. mapping to triangle_area : $\mathbb{R} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$:

```
>>> 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
```

Type hints are **not enforced** — they exist purely as documentation to make code more readable, and Python will not stop you calling a function with the "wrong" types.

Docstrings

A **docstring** (`"""..."""` immediately under the `def` line) documents what a function does and its preconditions, and gives example calls (written like REPL input/output) that double as tests:

```
>>> 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
```

General template

```
def function_name(arg0: type, arg1: type, ...) -> type:
    """
    Short description of the function for documentation.
    Preconditions (if any).
    >>> function_name(x, y, ...)
    expected output
    """
    ...
    function body
```

```
    ...
    return
```

See python-pep8-style-guide[1] for the naming, spacing, and line-length conventions used when writing functions like this.

[1] courses/csse1001/python-pep8-style-guide.pdf