# Introduction to Python

Chris Cornwell

September 2, 2025

## Outline

## Outline

## Assigning variables

- A variable is assigned by placing, on one line, `<variable name> = <assigned value>`.

```python
# The lines below assign three variables: x, y, and name_full
x = 5.11
y = 5
name_full = 'Chris Cornwell'
```

## Assigning variables

- A variable is assigned by placing, on one line, `<variable name> = <assigned value>`.

```python
# The lines below assign three variables: x, y, and name_full
x = 5.11
y = 5
name_full = 'Chris Cornwell'
```

- “Commenting out”, done by starting line with #.

## Assigning variables

- A variable is assigned by placing, on one line, `<variable name> = <assigned value>`.

```python
# The lines below assign three variables: x, y, and name_full
x = 5.11
y = 5
name_full = 'Chris Cornwell'
```

- "Commenting out", done by starting line with #.
- Possible to assign more than one variable in one line.

```python
x, y = 5.11, 5
# or, you could use
x = 5.11; y = 5
```

## Data type

Each variable has a *data type* (or, simply *type*). [1]

```python
x = 5.11
y = 5
name_full = 'Chris Cornwell'
```

↑ the types of the assigned vars are **`float`**, **`int`**, and **`str`** respectively.

[1] Information from Python documentation about standard types.

## Data type

Each variable has a *data type* (or, simply *type*). [1]

```python
x = 5.11
y = 5
name_full = 'Chris Cornwell'
```

↑ the types of the assigned vars are **`float`**, **`int`**, and **`str`** respectively.

Unlike in other programming languages, don't need to declare the types of the variables. Python *interprets* it. (The type might even *change* at a later point.)

[1] Information from Python documentation about standard types.

## Data type

Each variable has a *data type* (or, simply *type*). [1]

```python
x = 5.11
y = 5
name_full = 'Chris Cornwell'
```

↑ the types of the assigned vars are **`float`**, **`int`**, and **`str`** respectively.

Unlike in other programming languages, don't need to declare the types of the variables. Python *interprets* it. (The type might even *change* at a later point.)

- type **`int`**: like an integer.
- type **`float`**: like a real number in decimal form ...*kind of*.

[1] Information from Python documentation about standard types.

## Data type

Each variable has a *data type* (or, simply *type*). [1]

```python
x = 5.11
y = 5
name_full = 'Chris Cornwell'
```

↑ the types of the assigned vars are `float`, `int`, and `str` respectively.

Unlike in other programming languages, don't need to declare the types of the variables. Python *interprets* it. (The type might even *change* at a later point.)

- type `int`: like an integer.
- type `float`: like a real number in decimal form ...*kind of*.
- type `str`: a "string," or sequence of *characters* (that can be typed from keyboard).

[1] Information from Python documentation about standard types.

## Outline

## Numerical types

The four main operations[2] `+`, `-`, `*`, and / work as you would expect on numerical types `int` and `float`.

[2] Representing addition, subtraction, multiplication, and division.

## Numerical types

The four main operations[2] `+`, `-`, `*`, and / work as you would expect on numerical types `int` and `float`.

**Assigning after an operation.** Often want to change a variable by some amount (*e.g.*, increase it by 1), and keep the new value.

```python
# Compute y+1, then assign that to y
y = y + 1
# A convenient shorthand for line above is
y += 1
```

[2] Representing addition, subtraction, multiplication, and division.

## Numerical types

The four main operations[2] `+`, `-`, `*`, and / work as you would expect on numerical types `int` and `float`.

**Assigning after an operation.** Often want to change a variable by some amount (*e.g.*, increase it by 1), and keep the new value.

```python
# Compute y+1, then assign that to y
y = y + 1
# A convenient shorthand for line above is
y += 1
```

Equation above is an assignment, not mathematical. The shorthand works for other operations (like `-`, `*`, etc).

[2] Representing addition, subtraction, multiplication, and division.

## Numerical types

The four main operations[2] `+`, `-`, `*`, and / work as you would expect on numerical types `int` and `float`.

**Assigning after an operation.** Often want to change a variable by some amount (*e.g.*, increase it by 1), and keep the new value.

```python
# Compute y+1, then assign that to y
y = y + 1
# A convenient shorthand for line above is
y += 1
```

Equation above is an assignment, not mathematical. The shorthand works for other operations (like `-`, `*`, etc).

**Other operations.**

`**`, raises to a power; so, `y**2` has value $y^2$.

[2] Representing addition, subtraction, multiplication, and division.

## Numerical types

The four main operations[2] `+`, `-`, `*`, and / work as you would expect on numerical types `int` and `float`.

**Assigning after an operation.** Often want to change a variable by some amount (*e.g.*, increase it by 1), and keep the new value.

```
# Compute y+1, then assign that to y
y = y + 1
# A convenient shorthand for line above is
y += 1
```

Equation above is an assignment, not mathematical. The shorthand works for other operations (like `-`, `*`, etc).

**Other operations.**

`**`, raises to a power; so, `y**2` has value $y^2$.

% (called "mod"), finds the remainder; so, `5%3` is `2` and `6%3` is `0`.

[2] Representing addition, subtraction, multiplication, and division.

## Outline

## Basics of lists

`list` is a *sequential* data type in Python – it holds a sequence of “items.”

- Each item could be an `int`, each could be a `list`; possibly, can have some with one type, some with another.

## Basics of lists

`list` is a *sequential* data type in Python – it holds a sequence of “items.”

- Each item could be an `int`, each could be a `list`; possibly, can have some with one type, some with another.
- Example below: a list with `int` and `str` type items (and an empty list).

```python
my_list = [2, 3, 5, 'p']
empty_list = []
```

## Basics of lists

`list` is a *sequential* data type in Python – it holds a sequence of “items.”

- Each item could be an `int`, each could be a `list`; possibly, can have some with one type, some with another.
- Example below: a list with `int` and `str` type items (and an empty list).

```python
my_list = [2, 3, 5, 'p']
empty_list = []
```

Referring to a list item by index:

`my_list[0]` is `2` above, `my_list[1]` is `3`, and so on.

## Basics of lists

`list` is a *sequential* data type in Python – it holds a sequence of "items."

- Each item could be an `int`, each could be a `list`; possibly, can have some with one type, some with another.
- Example below: a list with `int` and `str` type items (and an empty list).

```
my_list = [2, 3, 5, 'p']
empty_list = []
```

Referring to a list item by index:
`my_list[0]` is `2` above, `my_list[1]` is `3`, and so on.

The + operation is defined on lists. It results in the *concatenation* of the lists – putting them together, end to end.

```
# the code below outputs [2, 3, 5, 'p', 11, 13]
my_list + [11, 13]
```

## Other operations on lists

- Multiplication by an integer: adds that many copies of the list together. For example, `[1,2]*3` will result in `[1,2,1,2,1,2]`, since

  `[1,2] + [1,2] + [1,2] = [1,2,1,2,1,2]`.

## Other operations on lists

- Multiplication by an integer: adds that many copies of the list together. For example, `[1,2]*3` will result in `[1,2,1,2,1,2]`, since

```
[1,2] + [1,2] + [1,2] = [1,2,1,2,1,2].
```

- Length of a list: use the function `len()`, with your list as input, to get the number of items in your list.

## Other operations on lists

- Multiplication by an integer: adds that many copies of the list together. For example, `[1,2]*3` will result in `[1,2,1,2,1,2]`, since

  `[1,2] + [1,2] + [1,2] = [1,2,1,2,1,2]`.

- Length of a list: use the function `len()`, with your list as input, to get the number of items in your list.
- Checking if an item is in a list: use the *keyword* `in` to check this. For example, if `my_list` is `[2, 3, 5, 'p']` then the first line below would result in `True`, the second would be `False`.

```
print( 2 in my_list )
print( 4 in my_list )
```

## `f`-strings

To produce an output string that uses values of some variables in memory, the best approach is to use what are called `f`-strings: in the string, a variable between `'{}'` will print out its value.

## f-strings

To produce an output string that uses values of some variables in memory, the best approach is to use what are called `f`-strings: in the string, a variable between `'{}'` will print out its value.

Say a variable `i` is in memory, with `i = 2`.

```python
# the next line will print out 'The value of i is 2.'
print(f'The value of i is {i}.')
```

## f-strings

To produce an output string that uses values of some variables in memory, the best approach is to use what are called `f`-strings: in the string, a variable between `'{}'` will print out its value.

Say a variable `i` is in memory, with `i = 2`.

```python
# the next line will print out 'The value of i is 2.'
print(f'The value of i is {i}.')
```

- It doesn't have to be the variable only. Could put something like `'{3*i}'` and Python will compute the value and print that.
- *Escape characters* can be handled inside strings also: e.g., `'\t'` will produce a tab; `'\n'` produces a newline.

## Outline

## Basic functions

Similar to most programming languages, Python uses *functions*, each of which takes some number of *arguments* (sometimes an argument is optional).

## Basic functions

Similar to most programming languages, Python uses *functions*, each of which takes some number of *arguments* (sometimes an argument is optional).

- Common function, already encountered: `print()`.

## Basic functions

Similar to most programming languages, Python uses *functions*, each of which takes some number of *arguments* (sometimes an argument is optional).

- Common function, already encountered: `print()`.
- Absolute value: `abs()`. Takes a numeric argument – either **`int`** or **`float`** type.

## Basic functions

Similar to most programming languages, Python uses *functions*, each of which takes some number of *arguments* (sometimes an argument is optional).

- Common function, already encountered: `print()`.
- Absolute value: `abs()`. Takes a numeric argument – either **`int`** or **`float`** type.
- Round to nearest integer: `round()`. Takes argument that is **`float`**. (If passed an **`int`** argument, will simply return it.)
    - Optional 2nd argument, the number of decimal digits.

## Basic functions

Similar to most programming languages, Python uses *functions*, each of which takes some number of *arguments* (sometimes an argument is optional).

- Common function, already encountered: `print()`.
- Absolute value: `abs()`. Takes a numeric argument – either **`int`** or **`float`** type.
- Round to nearest integer: `round()`. Takes argument that is **`float`**. (If passed an **`int`** argument, will simply return it.)
  - Optional 2nd argument, the number of decimal digits.

In Python, run the following to see how `round()` works.

```python
a = -3**2/8
print( a+8 )
print( (round(a+8), round(a+8, 2)) )
```

## Basic list methods

Methods are functions that you call on an instance of a class. There are several methods for lists. Here are two.

## Basic list methods

Methods are functions that you call on an instance of a class. There are several methods for lists. Here are two.

- `append()`: the command `my_list.append(x)` puts `x` at the end of `my_list`.
    - Changes `my_list` “in place.”

## Basic list methods

Methods are functions that you call on an instance of a class. There are several methods for lists. Here are two.

- `append()`: the command `my_list.append(x)` puts `x` at the end of `my_list`.
  - Changes `my_list` “in place.”
  - Is equivalent to `my_list += [x]`.

## Basic list methods

Methods are functions that you call on an instance of a class. There are several methods for lists. Here are two.

- `append()`: the command `my_list.append(x)` puts `x` at the end of `my_list`.
    - Changes `my_list` "in place."
    - Is equivalent to `my_list += [x]`.
- `remove()`: the command `my_list.remove(x)` takes out the *first* item in `my_list` that is equal to `x`.
    - Changes `my_list` in place, making it shorter.

## Basic list methods

Methods are functions that you call on an instance of a class. There are several methods for lists. Here are two.

- `append()`: the command `my_list.append(x)` puts `x` at the end of `my_list`.
    - Changes `my_list` "in place."
    - Is equivalent to `my_list += [x]`.
- `remove()`: the command `my_list.remove(x)` takes out the *first* item in `my_list` that is equal to `x`.
    - Changes `my_list` in place, making it shorter.
    - `my_list.pop(i)` does something similar with item at index `i`, but also returns (has as output) that item.

More information on working with lists and other basic classes, like strings, tuples, sets, and dictionaries: Tutorial from the Python documentation.