PRG 107 – Python Programming Kristiania University College Lec2

By Huamin Ren

Huamin.ren@kristiania.no

Høyskolen Kristiania

Outline - Python language basics I

1. Language semantics
2. Scalar types
3. Variables and functions
4. Data and Expressions
5. Control flow

Outline - Python language basics I

1. Language semantics

2. Scalar types

3. Variables and functions

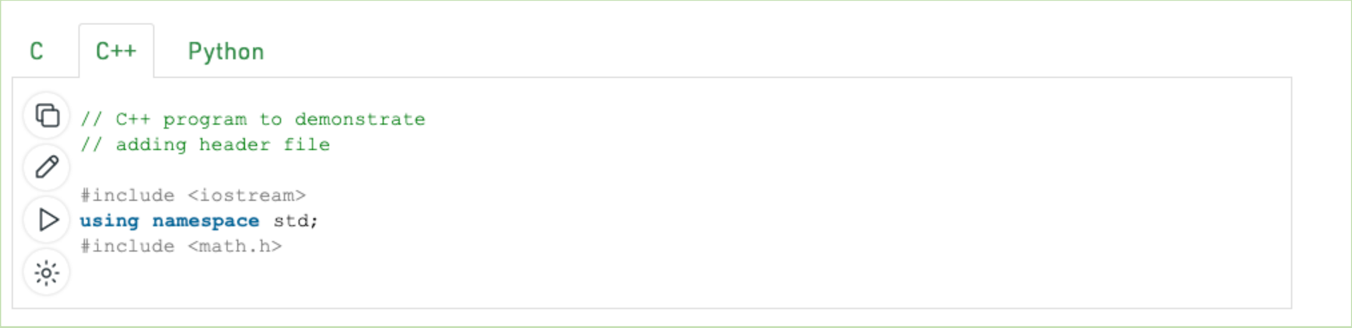
4. Data and Expressions

5. Control flow

1. Language Semantics

- The Python language design is distinguished by its emphasis on *readability*, *simplicity*, and *explicitness*. Some people go so far as to liken it to “executable pseudocode.”
- Indentation, not braces

C++ code:

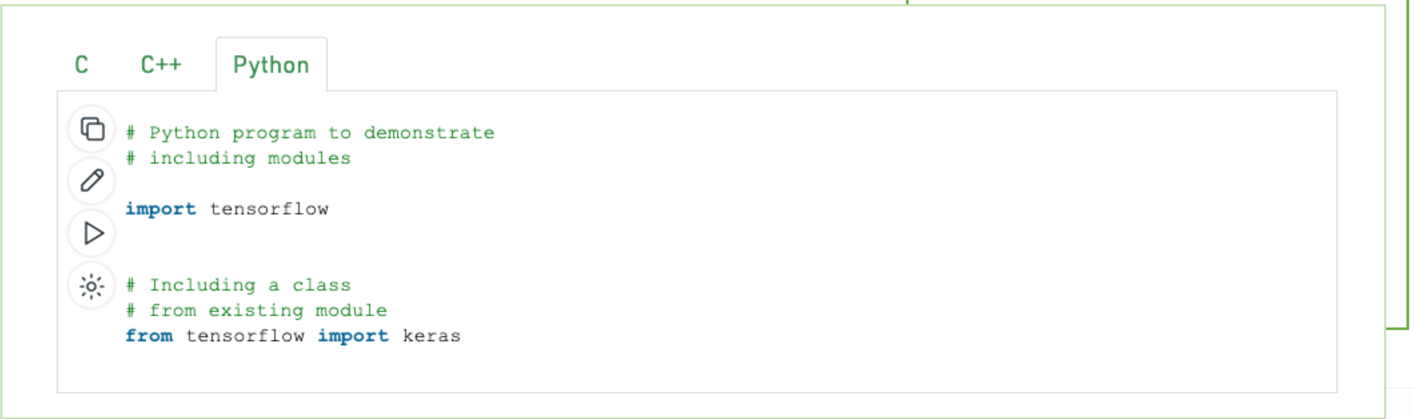


```
// C++ program to demonstrate
// adding header file

#include <iostream>
using namespace std;
#include <math.h>
```

1. Language Semantics

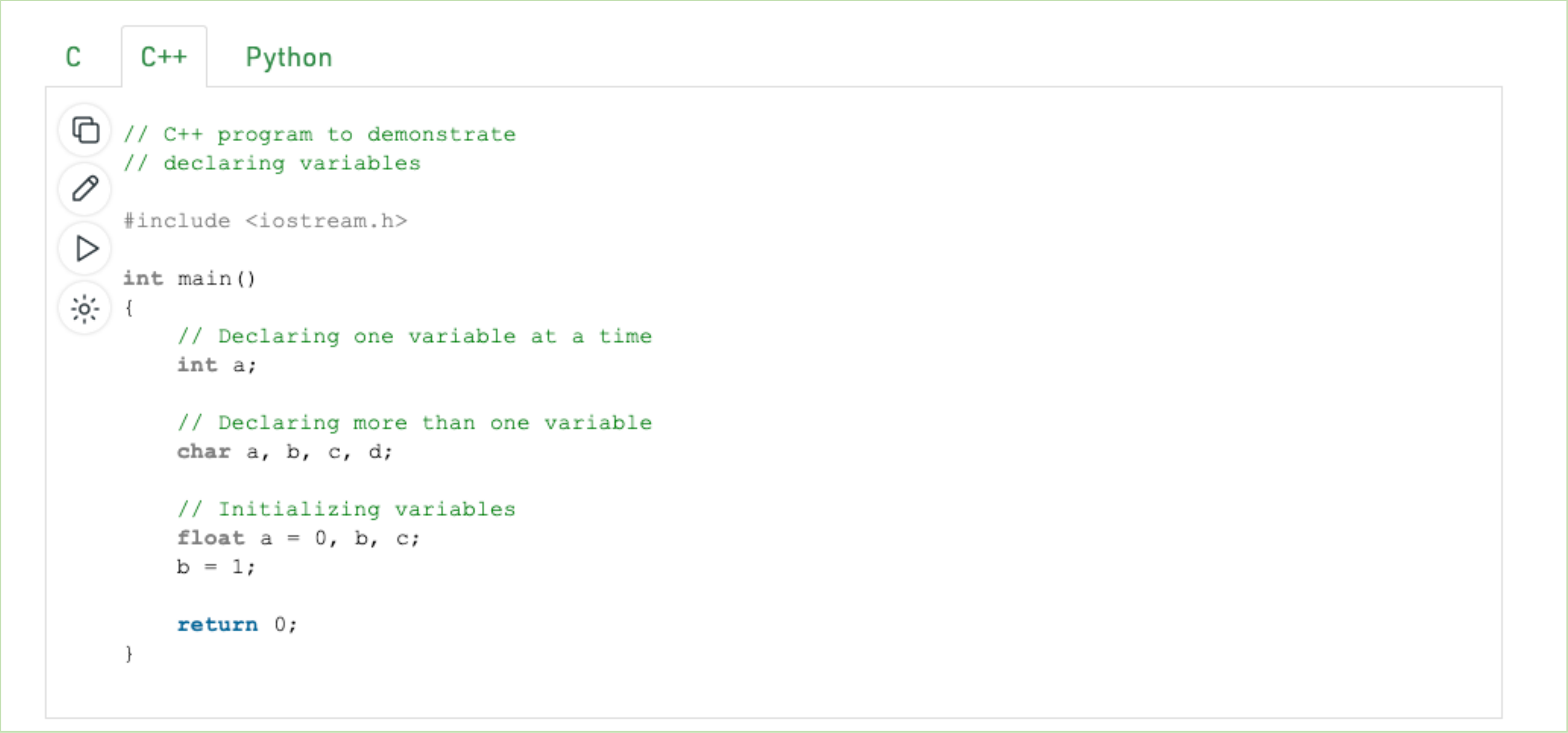
- The Python language design is distinguished by its emphasis on *readability*, *simplicity*, and *explicitness*. Some people go so far as to liken it to “executable pseudocode.”
- Indentation, not braces

Python code:



```
# Python program to demonstrate
# including modules

import tensorflow

# Including a class
# from existing module
from tensorflow import keras
```

1. Language Semantics

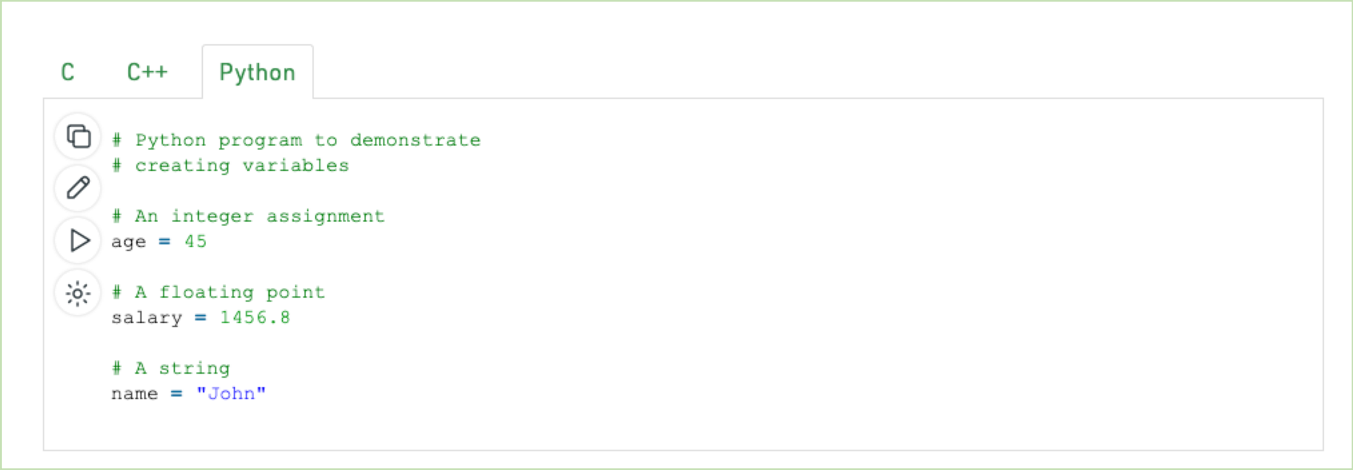
- The Python language design is distinguished by its emphasis on *readability*, *simplicity*, and *explicitness*. Some people go so far as to liken it to “executable pseudocode.”
- 

```
// C++ program to demonstrate
// declaring variables

#include <iostream.h>

int main()
{
    // Declaring one variable at a time
    int a;

    // Declaring more than one variable
    char a, b, c, d;

    // Initializing variables
    float a = 0, b, c;
    b = 1;

    return 0;
}
```

1. Language Semantics

- The Python language design is distinguished by its emphasis on *readability*, *simplicity*, and *explicitness*. Some people go so far as to liken it to “executable pseudocode.”
- Indentation, not braces

Python code:

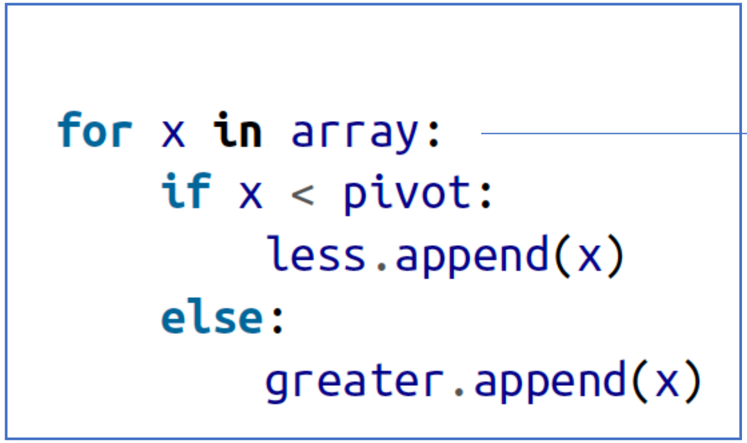


```
# Python program to demonstrate
# creating variables

# An integer assignment
age = 45

# A floating point
salary = 1456.8

# A string
name = "John"
```

1. Language Semantics

- The Python language design is distinguished by its emphasis on *readability*, *simplicity*, and *explicitness*. Some people go so far as to liken it to “executable pseudocode.”
- Indentation, not braces

PEP 8 -- Style Guide for Python Code:
https://www.python.org/dev/peps/pep-0008/



```
for x in array:
    if x < pivot:
        less.append(x)
    else:
        greater.append(x)
```

A colon:

A colon denotes the start of an indented code block after which all of the code must be indented by the same amount until the end of the block.

Semicolons?

- Python statements do not need to be terminated by semicolons.
- Semicolons can be used, however, to separate multiple statements on a single line

```
a = 5; b = 6; c = 7
```

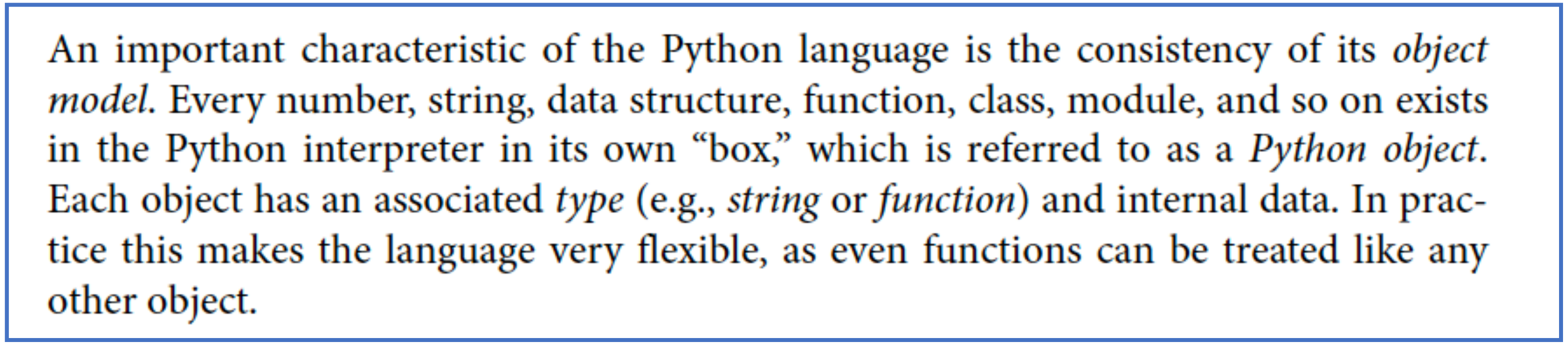
Or

```
a = 5;
b = 6;
c = 7
```

Or

```
a = 5
b = 6
c = 7
```

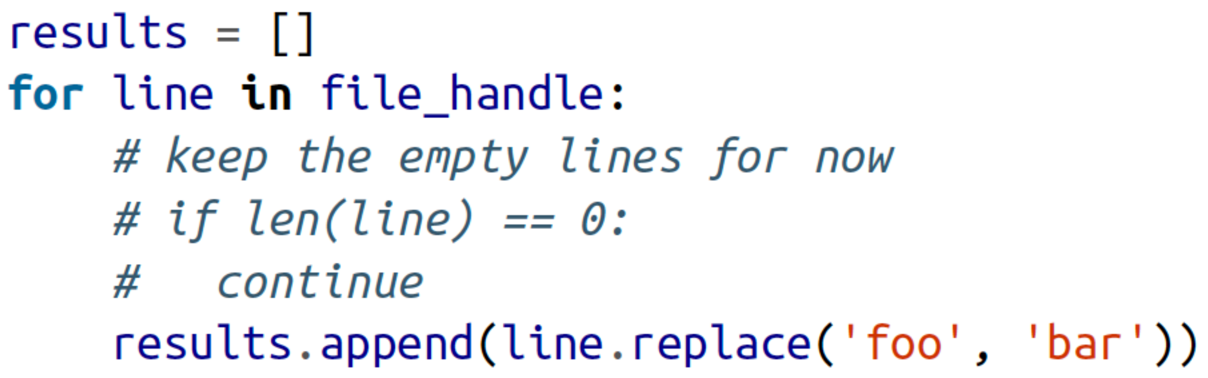
Everything is an object



An important characteristic of the Python language is the consistency of its *object model*. Every number, string, data structure, function, class, module, and so on exists in the Python interpreter in its own “box,” which is referred to as a *Python object*. Each object has an associated *type* (e.g., *string* or *function*) and internal data. In practice this makes the language very flexible, as even functions can be treated like any other object.

Comments

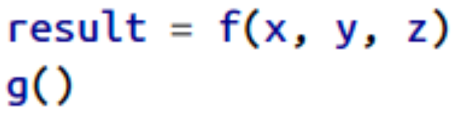
- Any text preceded by the hash mark (pound sign) # is ignored by the Python interpreter.
- However, for a large project, how to structure comments? Comment out the code



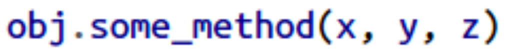
```
results = []
for line in file_handle:
    # keep the empty lines for now
    # if len(line) == 0:
    #   continue
    results.append(line.replace('foo', 'bar'))
```

Function and object method calls

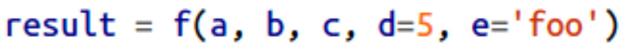
- Call functions using parentheses and passing zero or more arguments
- Optionally assigning the returned value to a variable



```
result = f(x, y, z)
g()
```



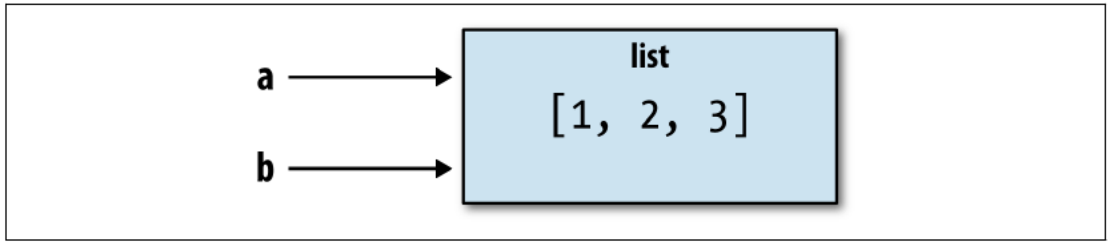
```
obj.some_method(x, y, z)
```

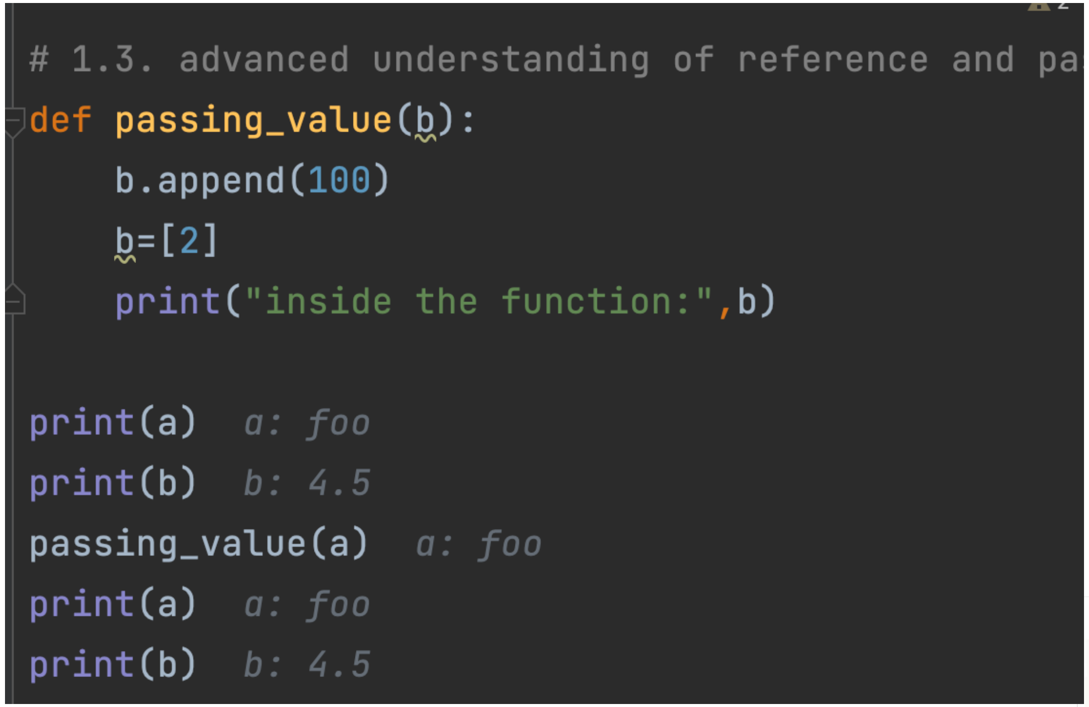


```
result = f(a, b, c, d=5, e='foo')
```

Almost every object in Python has attached functions, known as methods, that have access to the object's internal contents.

Variables and argument passing

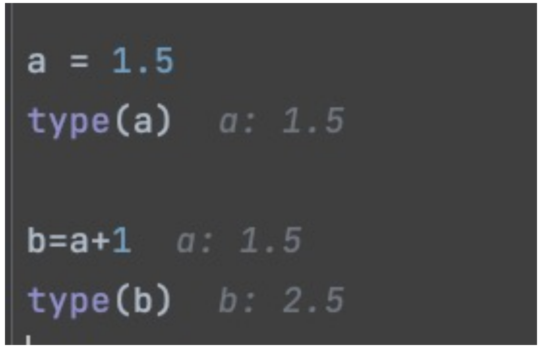




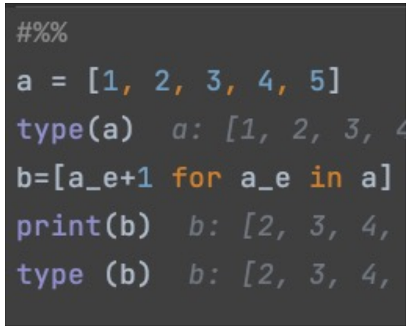
```
# 1.3. advanced understanding of reference and pa
def passing_value(b):
    b.append(100)
    b=[2]
    print("inside the function:",b)

print(a)   a: foo
print(b)   b: 4.5
passing_value(a)   a: foo
print(a)   a: foo
print(b)   b: 4.5
```

Dynamic references, strong types

- In contrast with many compiled languages, such as Java and C++, object references in Python have no type associated with them



```
a = 1.5
type(a)  a: 1.5

b=a+1  a: 1.5
type(b)  b: 2.5
```



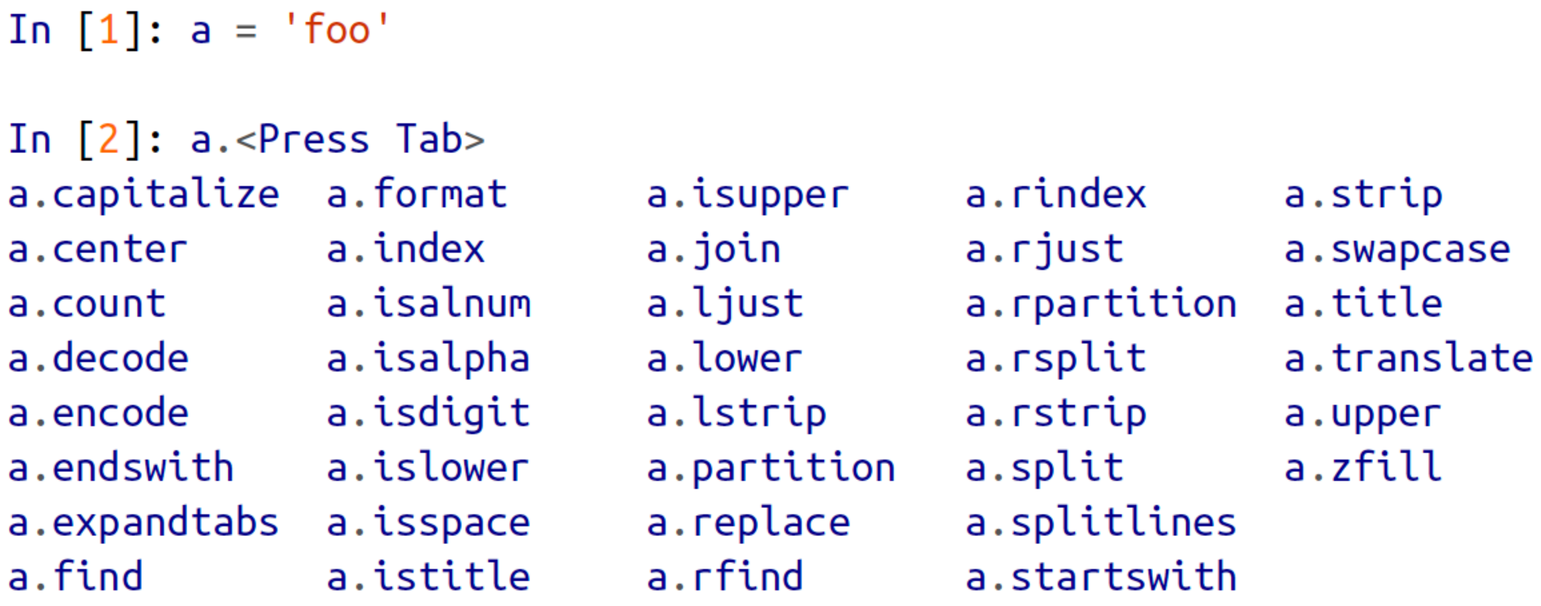
```
#%%
a = [1, 2, 3, 4, 5]
type(a)  a: [1, 2, 3, 4
b=[a_e+1 for a_e in a]
print(b)  b: [2, 3, 4,
type (b)  b: [2, 3, 4,
```

Check the type of the object

- Knowing the type of an object is important, and it’s useful to be able to write functions that can handle many different kinds of input. You can check that an object is an instance of a particular type using the isinstance function

Attributes and methods

- Objects in Python typically have both attributes (other Python objects stored “inside” the object) and methods (functions associated with an object that can have access to the object’s internal data).



```
In [1]: a = 'foo'

In [2]: a.<Press Tab>
a.capitalize  a.format      a.isupper     a.rindex      a.strip
a.center      a.index       a.join        a.rjust       a.swapcase
a.count       a.isalnum     a.ljust       a.rpartition  a.title
a.decode      a.isalpha     a.lower       a.rsplit      a.translate
a.encode      a.isdigit     a.lstrip      a.rstrip      a.upper
a.endswith    a.islower     a.partition   a.split       a.zfill
a.expandtabs  a.isspace     a.replace     a.splitlines
a.find        a.istitle     a.rfind       a.startswith
```

Imports

- In Python, a module is simply a file with the .py extension containing Python code.
- P36
- Import * from *

Binary operators and comparisons

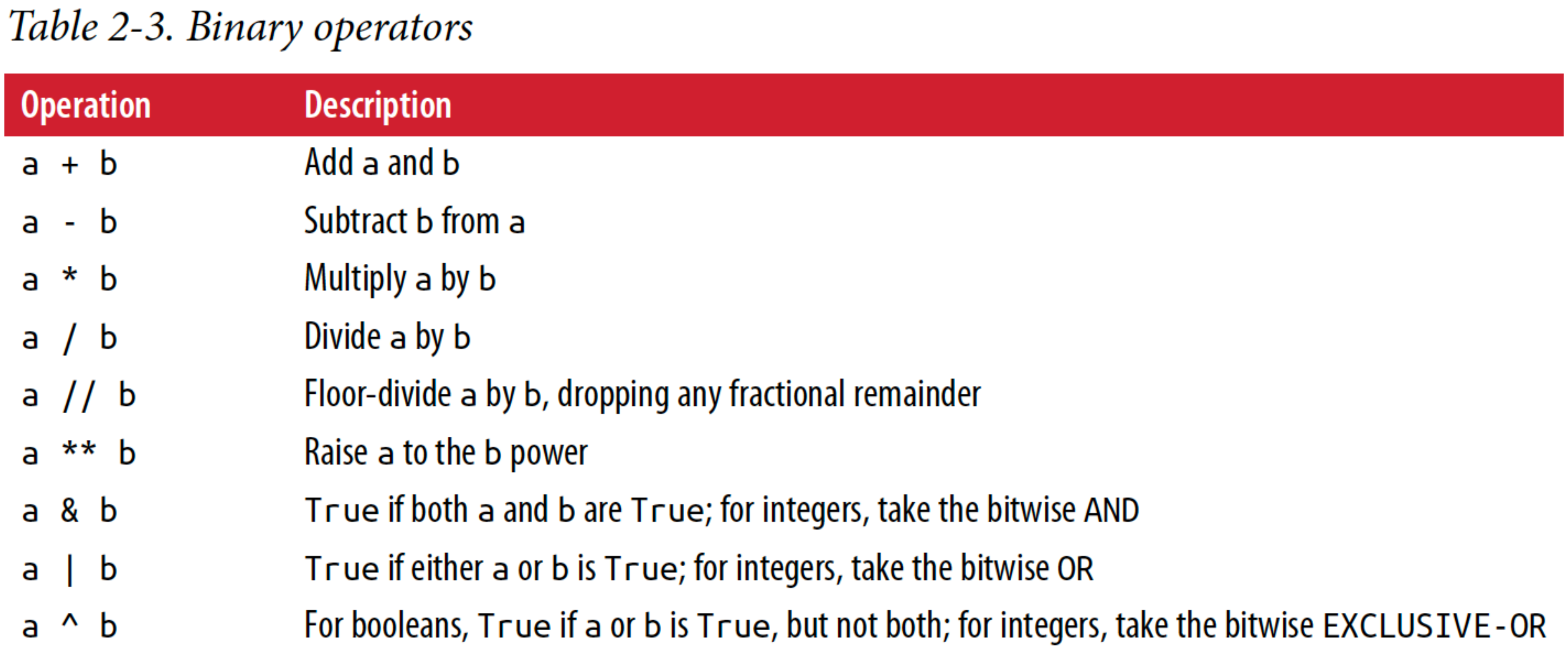
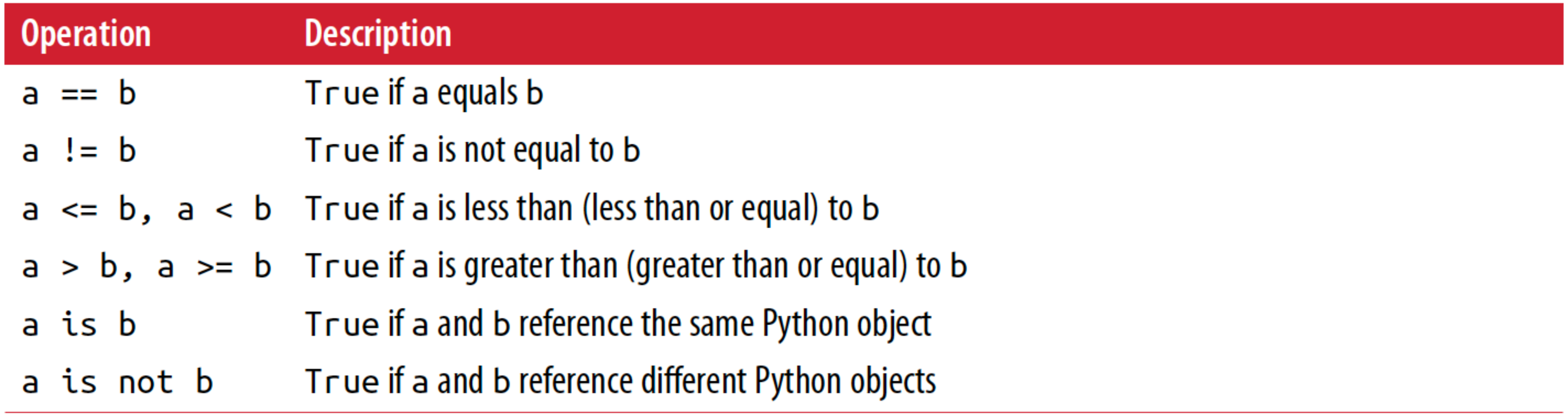


Table 2-3. Binary operators

Operation	Description
a + b	Add a and b
a - b	Subtract b from a
a * b	Multiply a by b
a / b	Divide a by b
a // b	Floor-divide a by b, dropping any fractional remainder
a ** b	Raise a to the b power
a & b	True if both a and b are True; for integers, take the bitwise AND
a | b	True if either a or b is True; for integers, take the bitwise OR
a ^ b	For booleans, True if a or b is True, but not both; for integers, take the bitwise EXCLUSIVE-OR



Operation	Description
`a == b`	`True` if a equals b
`a != b`	`True` if a is not equal to b
`a <= b, a < b`	`True` if a is less than (less than or equal) to b
`a > b, a >= b`	`True` if a is greater than (greater than or equal) to b
`a is b`	`True` if a and b reference the same Python object
`a is not b`	`True` if a and b reference different Python objects

Outline - Python language basics I

1. Language semantics

2. Scalar types

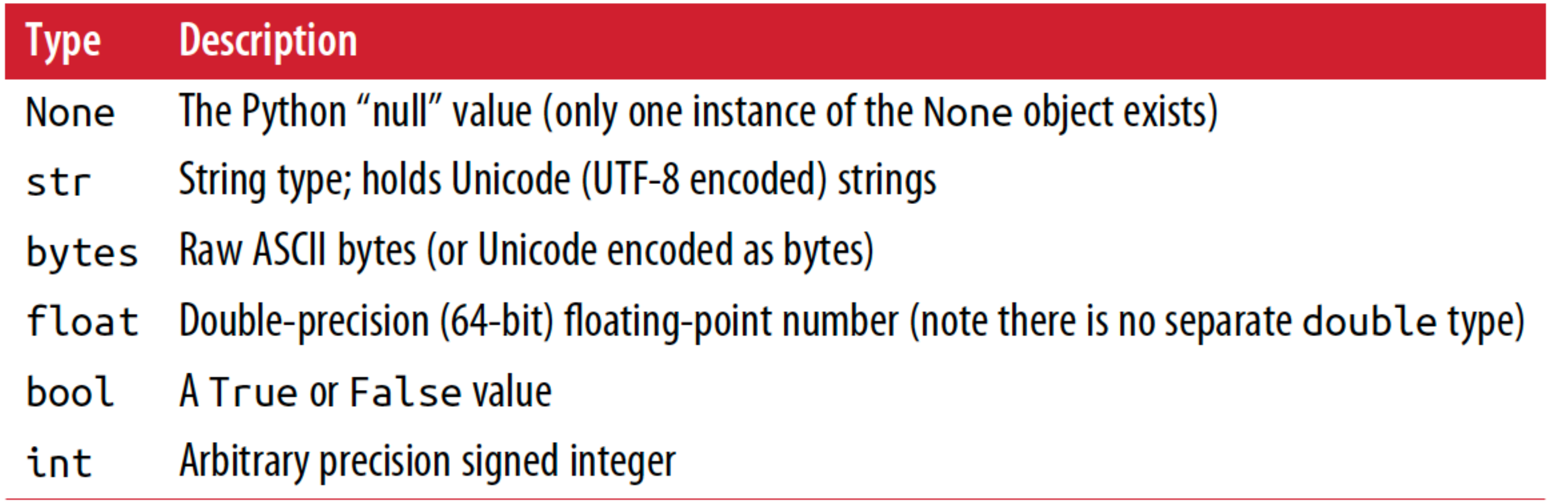
3. Variables and functions

4. Data and Expressions

5. Control flow

- A small set of built-in types for handling numerical data, strings, boolean (True or False) values, and dates and time.

Table 2-4. Standard Python scalar types



Type	Description
`None`	The Python "null" value (only one instance of the `None` object exists)
`str`	String type; holds Unicode (UTF-8 encoded) strings
`bytes`	Raw ASCII bytes (or Unicode encoded as bytes)
`float`	Double-precision (64-bit) floating-point number (note there is no separate `double` type)
`bool`	A `True` or `False` value
`int`	Arbitrary precision signed integer

2.1 Numerical types

- Int:

An int can store arbitrarily large numbers.

- Float: double-precision (64-bit) value

- How about division?

2.2 String

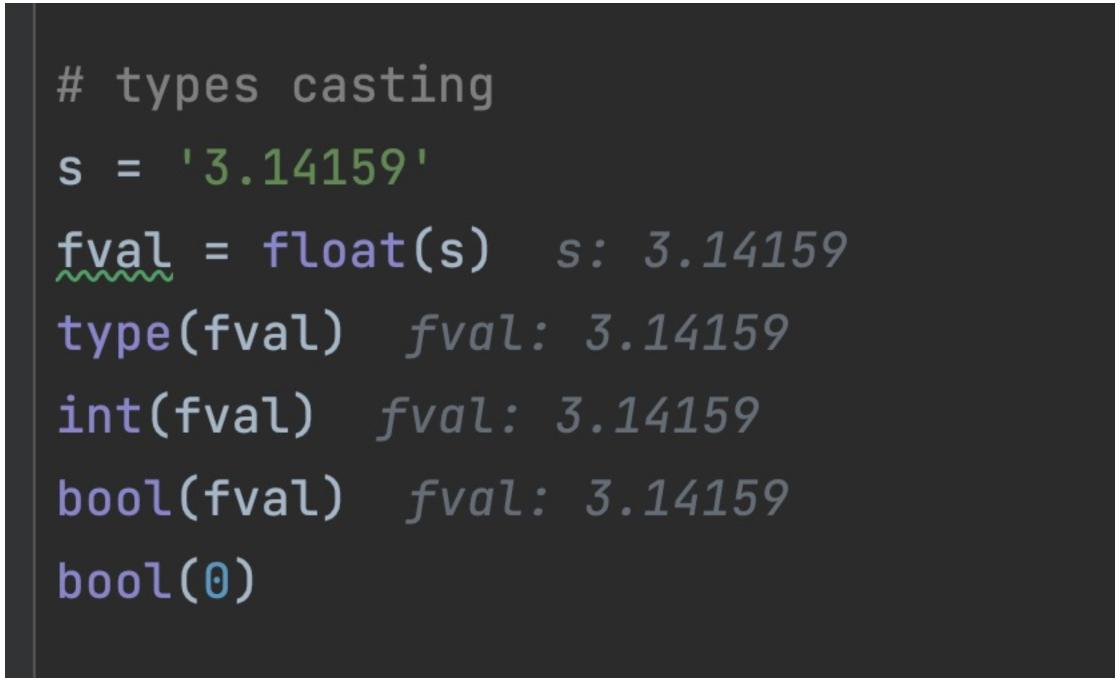
- Using single quotes ' or double quotes ”
- For multiline strings with line breaks, use triple quotes, either ''' or """”
- Python string is immutable
- Many python objects can be converted to a string using the str function
- The backslash character \ is an escape character, meaning that it is used to specify special characters like newline \n or Unicode characters.
- String +

2.3 Boolen

- True
- False

2.4 Type casting

- The str, bool, int, and float types are also functions that can be used to cast values to those types:



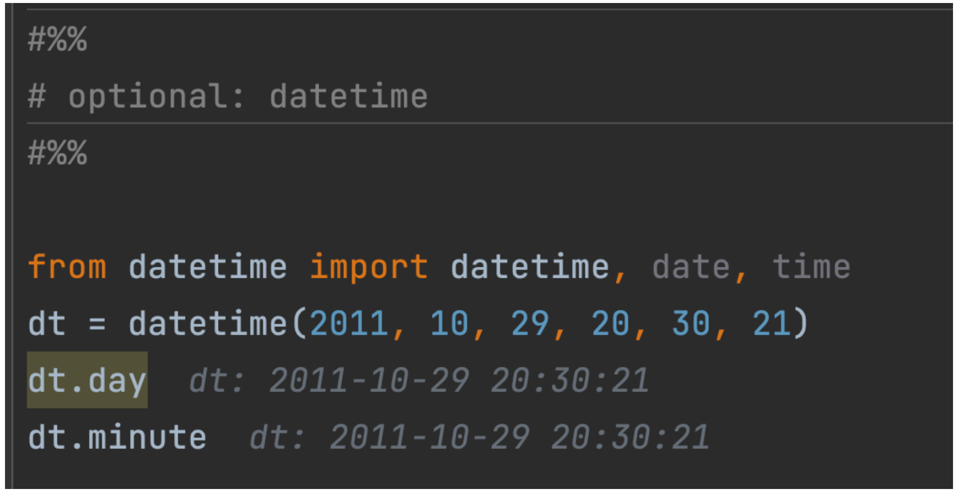
```
# types casting
s = '3.14159'
fval = float(s)  s: 3.14159
type(fval)  fval: 3.14159
int(fval)  fval: 3.14159
bool(fval)  fval: 3.14159
bool(0)
```

2.5 None

- None is the Python null value type. If a function does not explicitly return a value, it implicitly returns None
- Type(None)

2.6 Dates and Time (optional)

- The built-in Python datetime module provides datetime, date, and time types. The datetime type, as you may imagine, combines the information stored in date and time and is the most commonly used
- Strings can be converted (parsed) into datetime objects with the strptime function



```
#%%
# optional: datetime
#%%

from datetime import datetime, date, time
dt = datetime(2011, 10, 29, 20, 30, 21)
dt.day   dt: 2011-10-29 20:30:21
dt.minute   dt: 2011-10-29 20:30:21
```

2.7 Bytes and unicode

- Self-learning, P42.

Outline - Python language basics I

1. Language semantics
2. Scalar types
3. Variables and functions
4. Data and Expressions
5. Control flow

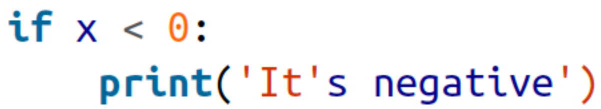
Outline - Python language basics I

1. Language semantics
2. Scalar types
3. ~~Variables and functions~~
4. ~~Data and Expressions~~
5. Control flow

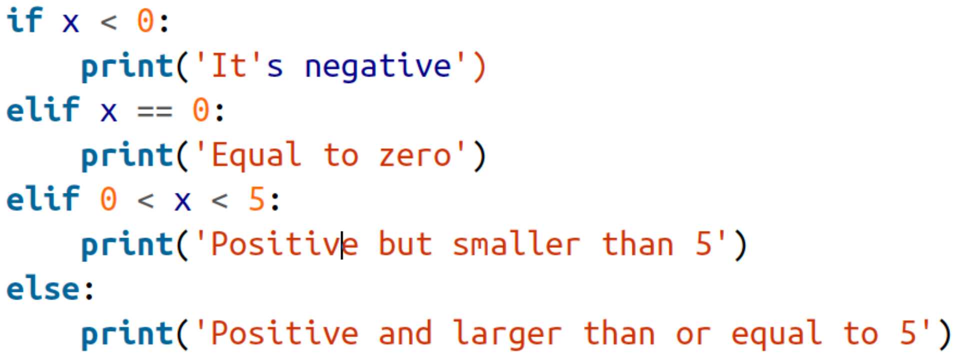
- Python has several built-in keywords for conditional logic, loops, and other standard control flow concepts found in other programming languages

if, elif, and else

- The if statement is one of the most well-known control flow statement types. It checks a condition that, if True, evaluates the code in the block that follows
- An if statement can be optionally followed by one or more elif blocks and a catchall else block if all of the conditions are False



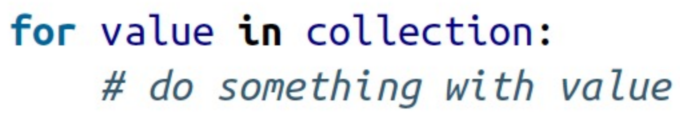
```
if x < 0:
    print('It's negative')
```



```
if x < 0:
    print('It's negative')
elif x == 0:
    print('Equal to zero')
elif 0 < x < 5:
    print('Positive but smaller than 5')
else:
    print('Positive and larger than or equal to 5')
```

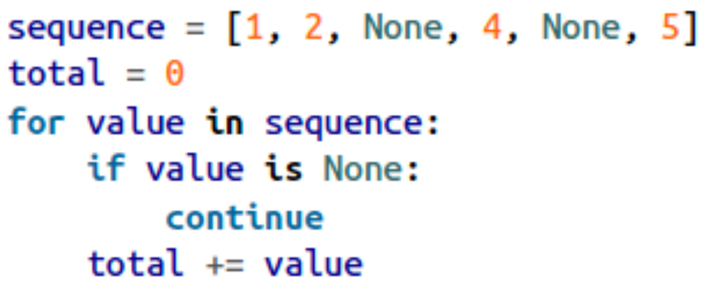
for loops

- for loops are for iterating over a collection (like a list or tuple) or an iterater. The standard syntax for a for loop is:



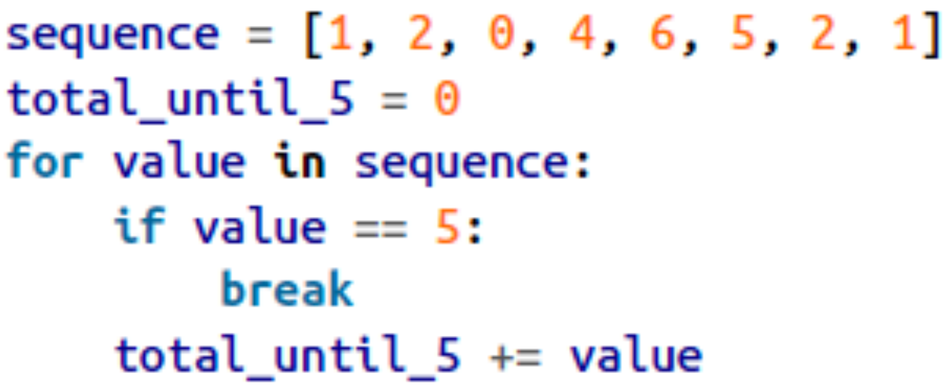
```
for value in collection:
    # do something with value
```

- Continue: you can advance a for loop to the next iteration, skipping the remainder of the block, using the continue keyword.



```python
sequence = [1, 2, None, 4, None, 5]
total = 0
for value in sequence:
    if value is None:
        continue
    total += value
```

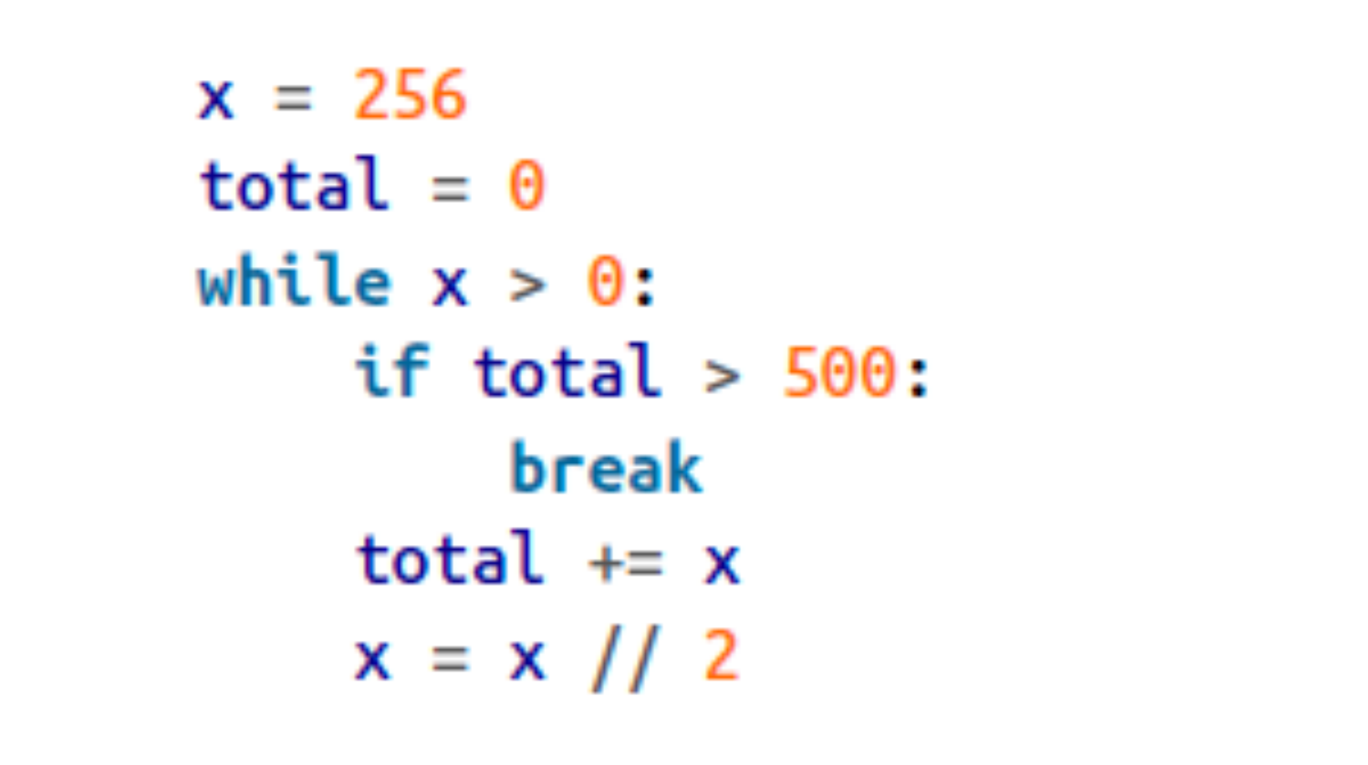
- Break: the break keyword only terminates the innermost for loop; any outer for loops will continue to run:



```
sequence = [1, 2, 0, 4, 6, 5, 2, 1]
total_until_5 = 0
for value in sequence:
    if value == 5:
        break
    total_until_5 += value
```

While

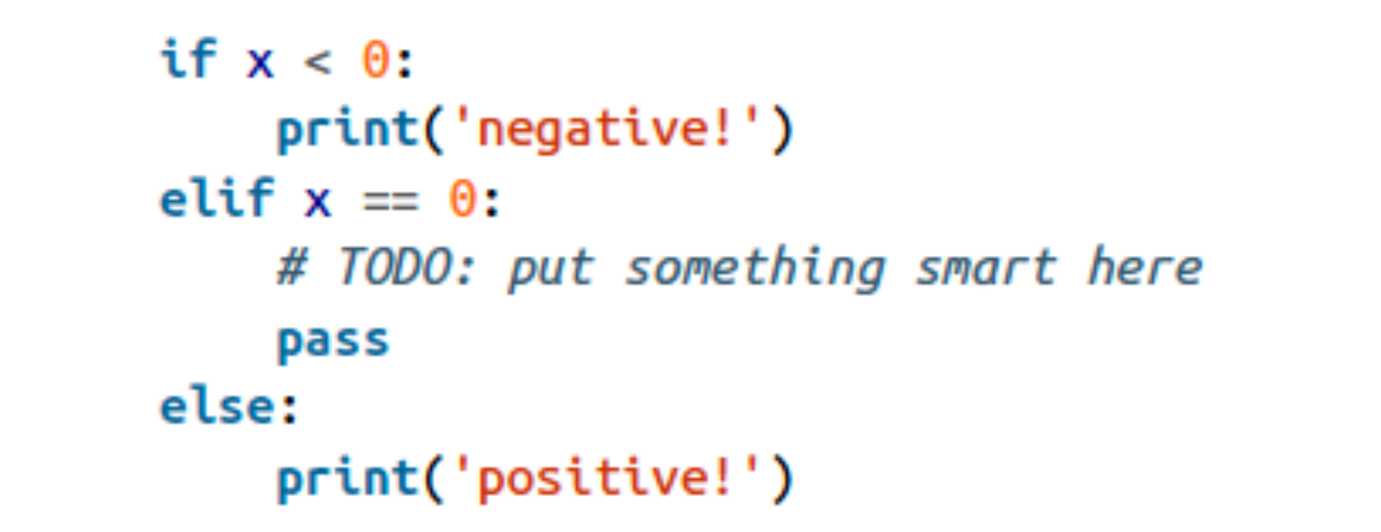
- A while loop specifies a condition and a block of code that is to be executed until the condition evaluates to False or the loop is explicitly ended with break:



```
x = 256
total = 0
while x > 0:
    if total > 500:
        break
    total += x
    x = x // 2
```

Pass

- pass is the “no-op” statement in Python. It can be used in blocks where no action is to be taken (or as a placeholder for code not yet implemented); it is only required because Python uses whitespace to delimit blocks:



```
if x < 0:
    print('negative!')
elif x == 0:
    # TODO: put something smart here
    pass
else:
    print('positive!')
```

Range

- The range function returns an iterator that yields a sequence of evenly spaced integers: