Introduction to Python

i0 Academy

Matthew Thompson 16/08/2023

What is Python?

- A high-level language, useful for many different programming tasks
- “The second best language for everything”
- Dynamically-typed and garbage collected
- Can be object-oriented or functional
- Easy to program, read and maintain code
- Favours indentation rather than use of parentheses (brackets)

What is Python used for?

- Web development (with Flask and Django frameworks, among others)
- Game development (with Pygame)
- Data Engineering and Data Science (with numpy, pandas and other libraries)
- Machine Learning (with pytorch, tensorflow, scikit-learn, etc)
- Also image processing, web scraping and many other uses

History of Python

- Created by Guido Van Rossum and released in 1991
- Python 2.0 released in 2000
- Python 3.0 released in 2008
- There was a long (and slightly painful!) transition from Python 2.0 to 3.0 from 2008 onwards, because version 3.0 was not backwards-compatible with 2.0
- Now Python 3.0 is most widely used, with 3.11 being the most recent stable release

VS Code setup and Hello World

Let's get set up with VS Code

- Install the Python extension for VS Code
- Create a directory called **python_intro** for our files
- Create **hello_world.py**, and type the code in snippet [1]
- Run it in the terminal with snippet [2]

[1]

```
print('Hello World!')
```

[2]

```
cd python_intro
python3 hello_world.py
```

- You can also run your python file with the VS Code command menu by typing **command-shift-P**, then finding the **Python: Run File in Terminal** command. It would be handy to add a keyboard shortcut for this!

Significant Whitespace in Python

```
def print_pattern(num_rows):
    for i in range(num_rows):
        for num_cols in range(num_rows-i):
            print("*", end="")
        print()
```

- To the left is an example of some Python code
- Don’t worry about understanding it yet!
- Note how few braces { } there are compared to other languages
- Instead, the indentation has meaning
- The convention is to use spaces to indent
- When you press **TAB**, most editors will automatically insert the correct number of spaces (same for deleting)

Functions

- Define a function with the **def** keyword, then a name, then parameters in brackets
- Don't forget to put a **:** after the parameters
- Then you must indent with the **TAB** key for the function body
- Return values using the **return** keyword

```
def add_numbers(num_a, num_b):
    return num_a + num_b

result = add_numbers(2, 3)
print(result)
```

- Here we're calling our function to add 2 and 3 together, then printing the result. Give it a try, and make some changes.

Comments, Variables and Scope

```
# everything after # is a comment

x = 1

def foo():
    y = 2

print(x) # this works
print(y) # this doesn't
```

```
def foo():
    global y
    y = 2

print(y) # this works now
```

- start comments with a **#**
- assigning a variable in Python is easy, just type the name, an =, and its value
- variables outside of functions have global scope (they can be used anywhere)
- variables inside functions have local scope (they can only be used inside the function)
- you can override this by defining a variable in a function with the **global** keyword (but don’t worry about this for now)

Exercise: hello_name

- Create a new file called **hello_name.py**
- Write a function that takes one parameter (a greeting), and prints the greeting, followed by a name
- Get the name by using the **input()** function to read in a name from the terminal (see snippet **[1]**)
- The output should look like snippet **[2]** (using whatever greeting you decide to use)

[1]

```
# get input from the terminal
name_str = input()
```

[2]

```
Enter your name:
Matt
Hello, Matt
```

Maths and Booleans

```
# Maths
1 + 1   # => 2
8 - 1   # => 7
10 * 2  # => 20
35 / 5  # => 7.0
7 % 3   # => 1
```

```
# Booleans
True   # => True
False  # => False
not True   # => False
not False  # => True
True and False  # => False
False or True   # => True
```

- Python has the usual maths operators like other languages
- Integers are whole numbers, floats are decimals
- Integers are coerced to floats on division
- The modulo **%** operator returns the remainder after division, and is surprisingly useful!
- Booleans are **True** and **False** (note the capital letters)
- Boolean operators are **and, not, or** (note the lower-case letters)

Conditionals (if/elif/else)

- Start a conditional clause with **if**, then a statement, then a colon **:**
- Remember: indentation has meaning in Python!
- The body of each clause is indented to the right
- Use **elif** for else if, and **else** for the default
- Both of these are at the same indentation level as **if**

```
if my_number > 10:
    print("Your number is bigger than 10.")
elif my_number < 10:   # This elif clause is optional.
    print("Your number is smaller than 10.")
else:                  # This is optional too.
    print("Your number is exactly 10.")
```

Exercise: sleep_in

```
sleep_in(False, False) # => True
sleep_in(True, False) # => False
sleep_in(False, True) # => True
```

- Write a function **sleep_in** with two parameters
- The first parameter is whether or not it's a weekday
- The second parameter is whether or not it's a holiday
- The function returns whether or not you can sleep in late that day
- If it's not a weekday or a holiday, it must be a weekend: you can sleep in! (function returns **True**)
- If it's a weekday but not a holiday, it returns **False** (you can't sleep in)
- If it's not a weekday and it's a holiday, it returns **True**, so you can sleep in

Lists

- Lists (or arrays) are useful when you want to order items or go through them one at a time
- You can mix different types within a Python list
- Append items on the end with **append()**
- Remove items with **pop(index)**
- Get an item by putting its index in square brackets: **my_list[3]**

```
my_list = [1, 'a', 4, 'hello', 'world', 6]
my_list.append('foo')
for item in my_list:
    print(item)
print('Item at index 3:', my_list[3])

'''
1
a
4
hello
world
6
foo
Item at index 3: hello
'''
```

Range function

```
my_list = range(10)
print(list(my_list))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

my_list2 = range(1,11)
print(list(my_list2))
# [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

my_list3 = range(0,26,5)
print(list(my_list3))
# [0, 5, 10, 15, 20, 25]
```

- To quickly create a list of consecutive numbers, you can use the **range()** function
- It takes a variable number of arguments
- **range(10)** generates a list from 0 to 9 (up to, not including 10)
- **range(1,11)** goes from 1 to 10
- **range(0,26,5)** goes from 0 to 25, at steps of 5
- **range(start, finish+1, step)**

Dictionaries

- Dictionaries (called “hash maps”, “lookup tables” or sometimes “objects” in other languages) are useful when you need to quickly look up a value, and order isn’t important
- They take the form of key-value pairs **{“key”: “value”}**
- Get a key’s value using square brackets: **dict[“key”]** or **dict.get(“key”)**
- You can also update it by passing in a new dictionary: **dict.update({“key”:”new-value”})**

```
dict = {"one": 1, "two": 2, "three": 3}
print(dict["one"]) # => 1
print(dict.get("two")) # => 2
dict.update({"three":33})
print(dict["three"]) # => 33
```

Loops

[1]

```
primes = [2, 3, 5, 7]
for prime in primes:
    print(prime)

print()

for n in range(len(primes)):
    print(primes[n])
```

=>

```
2
3
5
7

2
3
5
7
```

[2]

```
count = 0
while count < 5:
    print(count)
    count += 1  # count = count + 1
```

=>

```
0
1
2
3
4
```

- Use loops to go through lists one by one
- The two main types of loop in Python are **for** [1] and **while** [2] loops
- **for item in list:** is a shorthand that assigns each item to a variable (**item** in this case)
- otherwise, you can use **range(len(list))** to go through each index
- **while** loops keep on going until a condition is met, then stop when it becomes **True**

Exercise: fizz_buzz

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
```

- A classic interview question!
- Write a function that takes a number to go up to
- Then it counts up the numbers
- It prints “Fizz” if the number divides by 3
- It prints “Buzz” if the number divides by 5
- It prints “FizzBuzz” if the number goes into both 3 and 5
- It prints the number in all other cases
- See the example from 1 - 15 on the left
- The modulo operator **%** sure comes in handy here!

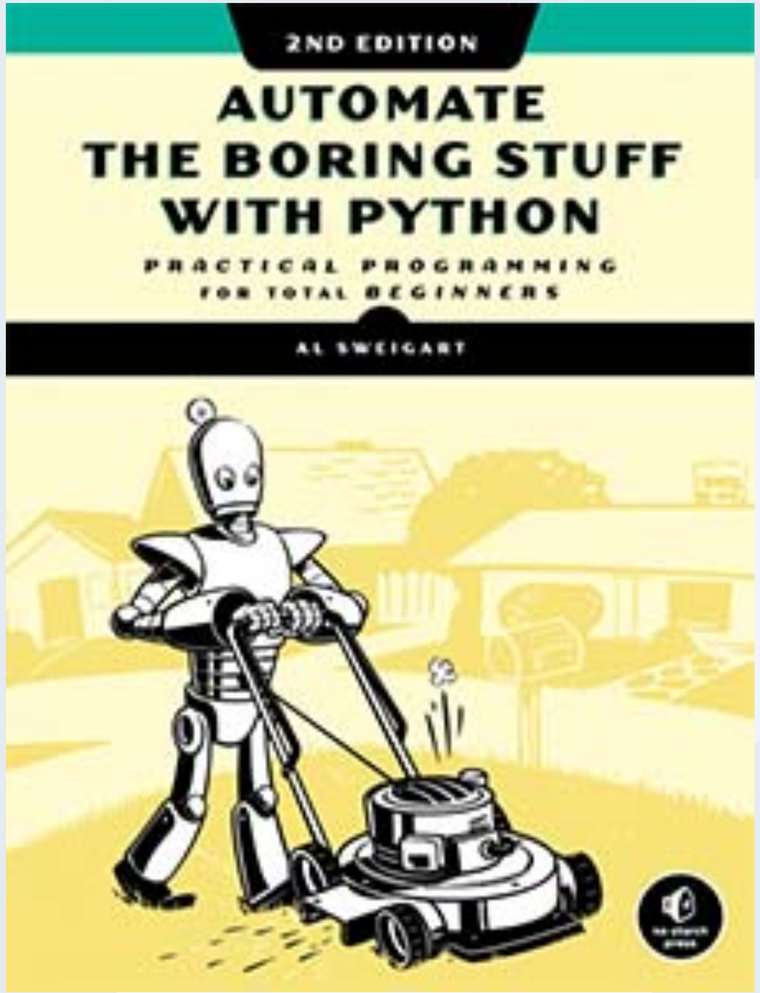
Exercise: star_triangle

- Write a program that takes a triangle width as input, then prints a triangle made out of star symbols: * (see right)
- The middle of the triangle is its width (in this case, 6 stars are printed there)
- Make use of all three range parameters: **range(start, stop, step)**
- Use **print('*', end='')** to print on the same line
- If this is too easy, try other shapes! Maybe a diamond?

```
Please enter triangle
width:
6

*
**
***
****
*****
******
*****
****
***
**
*
```

Python resources



- Automate the Boring Stuff with Python: https://automatetheboringstuff.com/
- Official Python docs: https://docs.python.org/3/
- The Hitch Hiker's Guide to Python: https://docs.python-guide.org/