Basic programming concepts

PGR Methods and Skills Bootcamp

Ana Matran-Fernandez & Viola Fanfani

26 November 2018

Institute for Analytics and Data Science
University of Essex

TODAY'S PLAN

- 9.00 – 10.45: Introduction to the Bootcamp
- 11.15 – 13.00: *Basic programming concepts*
- 14.00 – 17.00: Lab work
    - Set up accounts on Trello, Github, Slack
    - Python practice

TABLE OF CONTENTS

1. Introduction
2. Programming building blocks
3. Expressions, Conditional Statements and Loops
4. Functions, Lists and Strings
5. Dictionaries
6. Strings

Introduction

WHAT IS PROGRAMMING?

Programming is the act of creating and sending instructions to a computer. The computer will act on those instructions.

The instructions must be in a way that is understandable to the computer. You need to share a language with the computer. Python is one of those programming languages.

WHY PYTHON?

- Easy syntax
- Lots of documentation and libraries
- Great for data analysis
- Very widely used, so high chance that someone else had your problem and solved it.
    - Google it!
    - Stack Overflow: `https://stackoverflow.com/`

We will use Python3.

Programming building blocks

IDENTIFIERS

An identifier is a name used in a program to refer to a variable, class or function.

Identifiers may contain upper- or lower-case letters, digits and underscore characters. They must **not** begin with a digit.

It is good practice to assign meaningful names to the identifiers.

Examples:

```
x xyz counter age calculateAge i first_name
secondName j
```

VARIABLES

Variables are used to store data.

In Python there are 11 built-in types of data (although some libraries define their own types). They are: `int, float, complex, bool, str, list, tuple, range, dict, set, and frozenset`.

Numeric types: `int` (integers), `float` (real numbers), `complex` and `bool` (True/False, 1/0).

ASSIGNMENTS i

A variable assigns a name to a memory location.

To assign a value to a variable, we use the form:

```
variableName = value
```

E.g.:

```
age = 25
name = 'John'
distance = 33.5
```

ASSIGNMENTS ii

An assignment is used to store a value in a variable. Every time we do this, we overwrite the previous value that it contained.

E.g.:

```
>>> age = 20 + 5
>>> age = age + 1
```

What's the value of `age` now?

```
>>> age
```

FUNCTIONS i

Sometimes we want the computer to do the same thing multiple times, at different points of time.

Instead of re-typing the same lines of code again and again (which is inefficient and does not scale well), we can create a function and call it anywhere in our code.

A function is something you call (possibly with some parameters) to perform an action/a set of actions. It usually also returns one or more values.

FUNCTIONS ii

Functions in Python are created with the `def` statement:

```
def identifier(parameter1, ..., parameterN):
    ...
    actions
    ...
    return value
```

E.g.:

```
def hello(name):
    return 'Hello, ' + name + '!'

>>> print(hello('class'))
Hello, class!
```

COMMENTS

Comments make code more readable: they allow us to use our own language to explain the code. They are ignored by the computer.

Comments in Python start with a hashtag symbol: `#`

```
# This is a short comment.
```

They can also be defined by using `"""` at the beginning and end:

```
"""This is a long comment describing the full
functionality of my new function."""
```

CLASSES i

Python is an object-oriented programming language.

This means we can create objects (i.e., collections of data – attributes – with a set of methods for accessing and manipulating those data.

Most likely, you won’t use classes in this course, but you should know that they exist.

Expressions, Conditional Statements and Loops

COMPARISON OPERATORS

They are used to compare between two values.

- `a > b`: a is greater than b
- `a < b`: a is smaller than b
- `a == b`: a is equal to b (also for strings)
- `a != b`: a is different from b (also for strings)
- `a >= b`: a is greater or equal than b
- `a <= b`: a is smaller or equal than b

They evaluate to True or False.

Difference between = and ==:

= is used to assign a value to a variable.
== is used to check if two values are equal.

FLOW CONTROL: IF, ELIF, ELSE STATEMENTS

Flow control allows our program to execute different things depending on the instructions and conditions that we set.

Example:

```
myNumber = int(input('Write a number: '))
if myNumber > 400:
    print("The number is greater than 400.")
elif myNumber >= 300:
    print("The number is between 300 and 400.")
else:
    print("The number is smaller than 300.")
```

BOOLEAN OPERATORS

- `and`: `8>4 and 4>3` True
- `or`: `8>3 or 8>9` False
- `not`: `not True` False; `not 8>3` False

They evaluate to True or False.

WHILE LOOPS

Many times in programming, a block of code requires to be executed over and over as long as some condition holds True. It can run indefinitely until it meets or exceeds a certain condition.

E.g.,

```
number = -1
while number != 513:
     print('Type a number.')
     number = int(input())
print('Congratulations!')
```

This code won’t stop until you guess the right number.

While loops are good if you don’t know how many times you need to run the code.

FOR LOOPS i

Often, we know how many times the block of code will run.

E.g.,

```
for i in range(5):
    print(i ** 2)
```

What's the output?

```
total = 0
for num in range(20):
    total += num
print(total)
```

What's the output?

FOR LOOPS ii

```
all_reviews = [5, 4, 5, 3, 2, 5, 3, 2, 5, 4, 3, 1, 1, 2, 3, 5]
positive_reviews = []
for i in all_reviews:
      if i > 3:
            print('Pass')
            positive_reviews.append(i)
      else:
            print('Fail')
print(positive_reviews)
print(len(positive_reviews))
ratio = len(positive_reviews) / len(all_reviews)
percentage = ratio * 100
print('Percentage of positive reviews:', percentage, '%')
```

Functions, Lists and Strings

FUNCTIONS

When we create a function, we first define it (i.e., we give it a name) , we tell it what it should do (i.e., we create the body of the function) and we call it later.

Syntax:

```
def functionName(parameters):
     body of function
```

E.g.:

```
def even_check(num):
     if num % 2 == 0:
          print('Number is even.')
     else:
          print('Hmm, it is odd.')
even_check(50)
even_check(51)
```

PRINT FUNCTION

You’ve seen it in the examples before.

It can be used to check that our program runs correctly (by checking that the value/shape of a variable is what we expect).

E.g.:

```
>>> print('Printing a string')
>>> print(34/2)
>>> print('Number is', number)
```

LISTS

A list is a sequence of values.

Syntax:

```
myList = [item1, item2, item3, ..., itemN]
```

e.g.:

```
colors = ['red', 'green', 'blue']
heights = [1.78, 1.56, 1.65, 1.67, 1.5, 1.8, 1.63]
mixedList = ['egg', False, 400, 1.67, 'Ana']
```

ACCESSING VALUES IN A LIST

After we have a list, we will at some point want to access the values in it.

```
heights = [1.78, 1.56, 1.65, 1.67, 1.5, 1.8, 1.63]
```

Indexing starts at 0

`heights[0]` will show the first value: 1.78.
`heights[1]` = 1.56.
`heights[-1]` shows the last item: 1.63.

SLICING A LIST

What if we want to access multiple values? This is called slicing.

We use numbers and a colon to determine which part of the list to access: `n1:n2` selects all positions between `n1` and `n2-1` (included). If one of the numbers is missing:

- `:n2` is equivalent to `0:n2`
- `n1:` == all numbers from `n1` until the end (included)
    - Why not `n1:-1`?

```
integers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
print(integers[3:6])
print(integers[1:])
print(integers[:2])
print(integers[6:-1])
```

COMMON LIST OPERATIONS

- How many items in `myList`? `len(myList)`
- Extend a list: `myList.append("newItem")`
- Combine the values of multiple lists: `myList + newList`
- Change a value: `myList[0] = 'newValue'`
- Delete a value: `del myList[4]`
- Count appearances of an element: `myList.count("Pear")`
- Find first appearance of an element in the list: `myList.index("Pear")`
- Reverse elements of a list: `myList.reverse()`
- Sort elements of a list: `myList.sort()`
- Copy a list: `myList.copy()`

BEWARE!!

Imagine that `myList` is a list and you want to copy its contents to then modify it. You'd probably do `newList = myList`, right?

This won't work. If you now try to change a value of `newList`, it will also change in `myList`.

```
>>> myList = [1, 2, 3, 4, 5]
>>> newList = myList
>>> newList[2] = 44
>>> newList
[1, 2, 44, 4, 5]
>>> myList
[1, 2, 44, 4, 5]
```

Instead, you need to use the copy() method: `newList = myList.copy()`

Dictionaries

WHY USE DICTIONARIES?

In a list we group values and we can refer to them by **number** (position). In a dictionary, we can refer to each value by **name**.

Consider the example of a phone book. We could have two lists, one for names and one for numbers:

```
users = ['Laura', 'Martha', 'James', 'Martin']
numbers = [1324, 4365, 3333, 7890]
```

If we want to find Martha's number, we can do: `numbers[users.index('Martha')]`

This is not easily scalable. We'd much rather use something like `extensions['Martha']`, right?

So that's what a dictionary does:

```
extensions = {'Laura': 1324, 'Martha': 4365,
'James': 3333, 'Martin': 7890}
```

BASIC OPERATIONS WITH DICTIONARIES

- How many items in `phonebook`? `len(phonebook)`
- What's the number of person `person`? `phonebook[person]`
- Adding a new entry: `phonebook[newperson] = newNumber`
- Editing an existing entry: `phonebook[person] = newNumber`
- Remove an entry: `del phonebook[person]`
- Check if an entry exists: `person in phonebook` (it returns `True` or `False`)

We can also have dictionaries within dictionaries.

E.g., imagine that for each user we want to store their phone, age and address. How would you do this?

Strings

WHAT IS A STRING?

Strings, such as `'Hello, World!'` represent bits of text.

They’re delimited by `'` or `"` at the beginning and end.

You’ve already seen some examples in this course.

There is no difference between the single- and the double-quotation limits.

But it’s useful to have both: `"'It is late,' she said"`

`'"It\'s late," she said'` or `"'It\'s late,' she said"`

STRING METHODS i

Strings are immutable, so item and slice assignments are illegal

```
myList = "Boot"
myList[2] = "a" # We want to have myList = 'Boat'
```

Some possibly useful string methods for your project:

- How long is the string? `len(myString)`
- `myString.find("this")`
    - finds a substring within a larger string
    - returns the index where the substring starts
    - returns -1 if the substring is not found
- `myString.replace('original', 'new')`
    - returns a string where all the occurrences of 'original' have been replaced by 'new'
    - It's like a search and replace operation

STRING METHODS ii

- `mySString.split()`
    - splits a string into a sequence
    - by default, it splits where there’s a whitespace (new line, space, tab…)
    - but the splitter character can be specified: `myString.split("/")`
- `myList.strip()`
    - returns a string where the whitespaces at the beginning and end have been removed
    - You can also specify which characters to strip by listing them: `myString.strip("*/!")`

TODAY'S PLAN

- 9.00 – 10.45: Introduction to the Bootcamp
- 11.15 – 13.00: Basic concepts in programming
- 14.00 – 17.00: ***Lab work***
  - Set up accounts on Trello, Github, Slack, Overleaf
  - Python basics: practice in the lab