Automate the Boring Stuff

Who is this course for?

On its own, this book won’t turn you into a professional software developer any more than a few guitar lessons will turn you into a rock star. But if you’re an office worker, administrator, academic, or anyone else who uses a computer for work or fun, you will learn the basics of programming so that you can automate simple tasks such as these:

* Moving and renaming thousands of files and sorting them into folders
* Filling out online forms—no typing required
* Downloading files or copying text from a website whenever it updates
* Having your computer text you custom notifications
* Updating or formatting Excel spreadsheets
* Checking your email and sending out prewritten responses

These tasks are simple but time-consuming for humans, and they are often so trivial or specific that there’s no ready-made software to perform them. Armed with a little bit of programming knowledge, however, you can have your computer do these tasks for you. Example:

➊ passwordFile = open('SecretPasswordFile.txt')  
➋ secretPassword = passwordFile.read()  
➌ print('Enter your password.')  
   typedPassword = input()  
➍ if typedPassword == secretPassword:  
   ➎ print('Access granted')  
   ➏ if typedPassword == '12345':  
       ➐ print('That password is one that an idiot puts on their luggage.')  
  else:  
   ➑ print('Access denied')

You might not know anything about programming, but you could probably make a reasonable guess at what the previous code does just by reading it. First, the file SecretPasswordFile.txt is opened ➊, and the secret password in it is read ➋. Then, the user is prompted to input a password (from the keyboard) ➌. These two passwords are compared ➍, and if they’re the same, the program prints Access granted to the screen ➎. Next, the program checks to see whether the password is 12345 ➏ and hints that this choice might not be the best for a password ➐. If the passwords are not the same, the program prints Access denied to the screen ➑.

What is Python?

*Python* is a programming language (with syntax rules for writing what is considered valid Python code) and the Python interpreter software that reads source code (written in the Python language) and performs its instructions.

# Cheat Sheet

|  |  |
| --- | --- |
| **Command** | **Description** |
| Ctrl + [ / ] | Indentation on left / right respectively |
| // | Floor division e.x. 3 // 2 = 1 |
|  |  |
|  |  |
|  |  |

# Chapter 1 – Python Basics

REPL – Interactive shell which stands for Read Evaluate Print loop.

Graphical user interface, text, application

Description automatically generated

Figure 1 - First Python expression

2 + 2. 2 is a value while + is an operator. Expression will contain values and operators, which will result in an evaluation. Just typing 4 on it’s own and pressing Enter will also result in an evaluation, where 4 simply evaluates itself to 4.

Graphical user interface, application

Description automatically generated

Figure 2 - Math Operators from highest to lowest precedence

Precedence refers to the order of operations similar to mathematics.

## String Operations

**String example**: >>> ‘Hello, world!’

## String Concatenation and Replication

‘Alice’ + ‘Bob’ = AliceBob

In Python, it is possible to perform a **string replication** operation as follows:

‘Alice’ \* 5 will result in Alice printed five times.

## Variables

>>> spam = 40

>>> spam

40

A variable is *initialised* the first time a value is stored in it.

Graphical user interface, application

Description automatically generated

Figure 3 - Valid and Invalid Variable Names

Variables are case sensitive.

**Taking input and printing integers**

print('What is your age?')    # ask for their age  
   myAge = input()  
   print('You will be ' + str(int(myAge) + 1) + ' in a year.')

***Note***: int type cast variable to an integer for + 1 to be applied to it otherwise Python will think you are trying to concatenate with a string, as that is the only operation possible. Then the whole expression is converted into a string for concatenation to occur. Python gives an error because the + operator can only be used to add two integers together or concatenate two strings. You can’t add an integer to a string, because this is ungrammatical in Python. You can fix this by using a string version of the integer instead, as explained in the next section.

**Comment**

# This line is a comment

# Chapter 2 – Flow Control

Flow control statements can decide which Python instructions to execute under which conditions.

In Python, for **Boolean** variables, this is how you define their state, always start with capital T or F:

* spam = True
* apple = False

**Comparison operators** are exactly as the other languages. Even escape character \

Ex.

print('42 == \'42\' evaluates to ' + str(42 == '42'))

## Binary Boolean Operators

In Python, and or not Boolean operators are written as literals.

Ex. True and True

True

(4 < 5) and (9 < 6)

False

(1 == 2) or (2 == 2)

True

not True

False

## Blocks of Code

Grouping lines of code together is possible through indentation. There are three rules for blocks:

1. blocks begin when the indentation increases
2. blocks can contain other blocks
3. blocks end when the indentation decreases to zero or to a containing block’s indentation

## if statements

In Python the if statement is a bit different, rawer then other languages. Indentation plays an important part here. Syntax wise it is quite relaxed. *Also notice no semicolon anywhere.*

Ex.

If name == ‘Alice’:

Print(‘Hi Alice!’)

Elif age < 12:

Print(‘something else’)

Else:

Print(‘Unknown’)

## while statements

spam = 0  
while spam < 5:  
    print('Hello, world.')  
    spam = spam + 1 # spam+=1

continue statement

  while True:  
      print('Who are you?')  
      name = input()  
      if name != 'Joe':  
         continue  
       print('Hello, Joe. What is the password? (It is a fish.)')  
      password = input()  
       if password == 'swordfish':  
          break  
 print('Access granted.')

for loop and range function

print('My name is')  
for i in range(5):  
    print('Jimmy Five Times (' + str(i) + ')')

i is set to 0 and the value in range will go up to but not including this value. So 0-4

for I in range(12,16)

print(i)

result: 12,13,14,15

for I in range(0, 10, 2) will count zero to eight ( since 10 not inclusive) in intervals of 2

importing modules

import random, clock

**ending a program early - sys.exit()**

# Chapter 3 – Functions

## Defining a function

Def hello():

Print(‘Hello there’)

hello() # this will be a function call

helloUser(name)

# passing a value to the variable name, which make it an argument of the function helloUser()

**return keyword**

The return keyword allows the programmer to return a value from or the expression evaluation within a function. In Python, there is no need to declare what is expected as a return type for the defined function as can be seen below:

import random

def getAnswer(answerNumber):

       if answerNumber == 1:

           return 'It is certain'

       elif answerNumber == 2:

           return 'It is decidedly so'

       elif answerNumber == 3:

           return 'Yes'

       elif answerNumber == 4:

           return 'Reply hazy try again'

       elif answerNumber == 5:

           return 'Ask again later'

       elif answerNumber == 6:

           return 'Concentrate and ask again'

       elif answerNumber == 7:

           return 'My reply is no'

       elif answerNumber == 8:

           return 'Outlook not so good'

       elif answerNumber == 9:

           return 'Very doubtful'

r = random.randint(1, 9)

fortune = getAnswer(r)

print(fortune)

**None value**

None represents the absence of a value. The none value is the only value of the *Nonetype* data type. This is quite useful when you need to store something that won’t be confused for a real value in a variable.

**Note:** behind the scenes Python add a return None to the end of any function definition with no *return* statement. If the return statement is used without a value, then *None* is returned.

## The Call Stack

Similar to a long conversation with several details involving many subjects, calling a function does not send the execution on a one-way trip to the top of a function. Python will remember which line of code called the function so that the execution can return there when it encounters a *return* statement. If that original function called other functions, the execution would return to those functions first, before returning from the original function call.