



Learning to Program in
>> Python



PG ONLINE

PM Heathcote

Learning to Program in Python >>

P.M. Heathcote

Published by
PG Online Limited
The Old Coach House
35 Main Road
Tolpuddle
Dorset
DT2 7EW
United Kingdom
sales@pgonline.co.uk
www.pgonline.co.uk

2017



PG ONLINE

Graphics: Paul Raudner / PG Online Ltd

Design and artwork: PG Online Ltd

First edition 2017, reprinted June 2018

A catalogue entry for this book is available from the British Library

ISBN: 978-1-910523-11-7

Copyright © PM Heathcote, 2017

All rights reserved

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the prior written permission of the copyright owner.

Printed and bound in Great Britain

Preface

Programming is fun! Trial and error is to be encouraged and you should type all of the examples and try all the exercises to get used to entering and debugging programs and to see how the programs run.

Python is one of the most popular programming languages both in schools and in industry. Python is used by Google, NASA, Instagram, Facebook and thousands of other companies to develop their applications. It is easy to learn and is a suitable language for tackling projects from school onwards.

This book is intended for individuals and students who may have done some programming in other languages, but are not familiar with Python. It is intended that users of the book should work through the book sequentially, starting at Chapter 1. However, it will be a useful reference book for students on a programming course or anyone working on a programming project.

It teaches basic syntax and programming techniques, and introduces three built-in Python modules:

- **Tkinter**, used for building a graphical user interface, which is an option that some users may like to include in their project work.
- **SQLite**, which enables the creation and processing of a database from within a Python program. This provides an alternative to writing to a text file when data needs to be stored and retrieved.
- **pdb**, Python's debugging module, which can be used to help find elusive logic errors.

Questions and exercises are included throughout every chapter. Over 120 Python programs for all the examples and exercises given in the book may be downloaded from www.pgonline.co.uk. We strongly advise you to write your own code, and check your solutions against the sample programs provided.

Enjoy – the sky's the limit!

Downloading Python 3

Python is available to be downloaded free from <https://www.python.org/downloads/>. The version used in this book is version 3.6. All the programs have been written and tested in IDLE, Python's own integrated development environment. Many schools and individuals may prefer to use alternative development environments and the book is equally applicable to these.

Contents

Chapter 1 – Data types, operators and I-O	1
Programming in Python	1
Programming in interactive mode	1
Data types	2
Rounding a result	3
Naming objects	4
Augmented assignment operators	6
The print statement	6
The input statement	9
Chapter 2 – Strings and numbers	11
Script mode	11
Adding comments	12
Keeping the console window open	12
String methods	12
Syntax errors	13
Inputting numbers	14
Converting between strings and numbers	15
Functions and methods	16
Chapter 3 – Selection	17
Programming constructs	17
Boolean conditions	18
The elif clause	18
Nested selection statements	19
Complex Boolean expressions	19
Importing library modules	20

Chapter 4 – Iteration	22
The for loop	22
The while loop	24
String processing	25
Slicing strings	27
Interrupting execution	28
Chapter 5 – Lists and tuples	29
Python lists	29
Operations on lists	30
Appending to a list	31
List processing	32
Two-dimensional lists	33
Tuples	35
Chapter 6 – Validating user input	37
Validating user input	37
The ASCII code	38
Functions ord() and chr()	40
Regular expressions	41
Chapter 7 – Searching and sorting	44
Dictionary data structure	44
Storing a list in a dictionary	47
Sorting a list	47
Sorting a two-dimensional list	47
Chapter 8 – Functions	50
Types of subroutine	50
Built-in functions	50
Writing your own functions	51

Using parameters and return values	52
A note about procedures and functions	55
Local and global variables	56
Chapter 9 – Reading and writing files	59
Storing data	59
Records and fields	59
Opening, reading and closing a text file	60
Writing to a file	64
File processing	65
Formatting output	68
Chapter 10 – Databases and SQL	71
Flat file databases	71
Records, fields and primary keys	72
Querying a database	73
Adding a record to a database table	73
Updating a record	74
Deleting records from a table	74
Chapter 11 – Python’s SQLite module	76
Using SQL commands in a Python program	76
Creating a database	76
Importing data from a text file	78
Creating a new database and loading it with data	80
Querying the database	81
Adding records entered by the user	83
Trapping errors	84
Deleting a record	84
Updating the database	85

Chapter 12 – Introduction to Tkinter	88
The Python Tkinter module	88
The “Hello World” program	89
Widgets	90
Placing widgets in a window	91
Responding to user input	92
Setting window parameters	93
Chapter 13 – Developing an application using Tkinter	95
Sample Application 1	95
Designing the data input window	95
Building the form	96
Sample Application 2	102
Sample Application 3	106
Chapter 14 – Program design	110
Planning a program	110
The sample task	110
Chapter 15 – Testing and debugging	113
Drawing up a test plan	113
Python module pdb	115
Index	119

Chapter 1

Data types, operators and I-O

Objectives

- Run commands in interactive mode
- Use string, numeric and Boolean data types and operators
- Learn the rules and guidelines for naming variables
- Use input and output statements

Programming in Python

Python is a popular, easy-to-learn programming language. A Python program is simply a series of instructions written according to the rules or **syntax** of the language, usually designed to perform some task or come up with a solution to a problem. You write the instructions, and then the computer translates these instructions into binary machine code which the computer can execute. It will do this using a translator program, which could be either a **compiler** or an **interpreter**. Python uses elements of both an interpreter and a compiler.

Python comes with an **integrated development environment** called **IDLE** which enables you to enter your program, save it, edit it, translate it to machine code and run it once it is free of syntax errors. If you have written a statement wrongly, that will be reported by the interpreter as a syntax error, and you can correct it and try again.

Programming in interactive mode

Python has two modes of entering and running programs. In **interactive mode**, you can type instructions and Python will respond immediately. This is very useful for trying out statements that you are not sure about, and is a good place to start. However, you cannot save a program that you have written in interactive mode. This has to be done in **script mode**, described in the next chapter.

Chapter 2

Strings and numbers

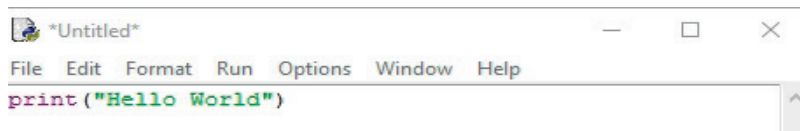
Objectives

- write and execute programs in script mode
- learn some useful string methods and functions
- convert string input to numerical values and vice versa
- identify and correct syntax errors

Script mode

In the last chapter we used **interactive mode**, which allows you to test out different Python statements and gives instant feedback. However, if you want to save a program so that you can load and run it later, you need to use **script mode**. Alternatively, you can use one of the many interactive development environments (IDEs) that are available to create, edit, save and run Python programs. In this book the screenshots show programs entered and executed in Python IDLE.

You can open an **Editor window** in Python's IDLE (integrated development environment) from the interactive Shell window. Select *File, New File* from the menu bar, and a new window will open. In this window, type a one-line program:



```
*Untitled*
File Edit Format Run Options Window Help
print("Hello World")
```

- Before you can run the program, you must save it, so select *File, Save* from the menu bar and save it with the name **hello world.py** in a suitable folder
- Then select *Run, Run Module* from the menu or use the shortcut key F5
- The interactive window will appear with the result:

```
Hello World
```



You can leave the interactive window open while you are writing programs in script mode. That way, if you are not sure about a statement, you can try it before writing it in your program.

Adding comments

It's a good idea to include comments at the top of every program (beyond the very trivial) that you write, giving the name and purpose of the program, your name and the date you wrote the program. You may also want to document in which folder you have saved it so you can quickly find it again after a few weeks or months.

Within the program, add comments to explain the purpose of any tricky bit of code and how it works.

To write a comment, type the # symbol. Anything to the right of # will be ignored.

Keeping the console window open

If you run the Python program by double-clicking its name in your folder, rather than launching it from IDLE, the program will run and then the console window will immediately close so that you can't see what happened. To keep it open, add a line to the end of your program:

```
input("\nPress Enter to exit: ")
```

The window will remain open until the user presses the **Enter** key.

2

String methods

Every data type in Python is an **object**. Strings, integers and floating point numbers are all objects, and they have built-in **methods** which perform useful tasks. Some useful string methods are shown in Table 2.1.

Method	Example	Description
upper	<code>astring.upper()</code>	returns astring all in uppercase
lower	<code>astring.lower()</code>	returns astring all in lowercase
index	<code>astring.index(item)</code>	returns the index of the first occurrence of item in astring, or an error if not found
find	<code>astring.find(item)</code>	returns the index of the first occurrence of item in astring, or -1 if not found
replace	<code>astring.replace(old,new)</code>	replaces all occurrences of old substring with new in astring

Table 2.1: String methods

```

File Edit Format Run Options Window Help
#Program name: Ch 2 Example 2 string methods.py
#Tests string methods

astring = "You've done a good job"
print("\nOriginal string:" astring)
print("\nUppercase string:", astring.upper)
print("\nNew String:", astring.replace("good", "excellent"))
print("\nOriginal string:", astring)

```

There is a missing comma before the variable name `astring`. You can click OK, correct the error, resave and try again.

Q1 There are two more errors in the program. Can you spot them?

Inputting numbers

A simple program to find the cost of lunch is shown below.

Example 3

```

#Program name: Ch 2 Example 3 Cost of lunch.py
main = 3.00
juice = 0.75
banana = 1.00
total = main + juice + banana
print("Total for lunch: ", total)

```

This prints out

```
Total for lunch: 4.75
```

Example 4

To make the program more useful, we could ask the user to enter the cost of the main course, juice and banana. Here is a first attempt:

```

#Program name: Ch 2 Example 4 Cost of lunch v2.py
main = input("Enter cost of main course: ")
juice = input("Enter cost of juice: ")
banana = input("Enter cost of banana: ")
total = main + juice + banana
print("Total for lunch: ", total)

```

Q1

Write Python code to do the following:

- Print the numbers 10 - 0 starting at 10. Then print "Lift-off!" Import the `time` module at the start of the program with the statement `import time`. Include a time delay of 1 second before printing each number, using the statement `time.sleep(1)`.
- Ask the user to enter 5 numbers. Keep a running total and print the total and average of the numbers.

The while loop

The `while` loop is used when the number of times the loop will be performed is initially unknown. The loop is performed as long as a specified Boolean condition is **True**. If the condition is **False** before the `while` statement is encountered, the loop will be skipped.

The Boolean condition is tested in the `while` statement at the start of the loop, and again each time the loop is completed. If it becomes **True** halfway through the loop, the remaining statements in the loop will be executed before the condition is re-tested.

Example 3

Write code to accept a series of integer test results from a user, find and print the maximum and minimum results. Assume all the results are between 0 and 100. The end of the input is signalled by an input of -1.

```
#Program name: Ch 4 Example 3 max and min.py
testResult = int(input("Please enter test result: "))
# Set maximum and minimum to first test result
maxResult = testResult
minResult = testResult

while testResult != -1:
    if testResult > maxResult:
        maxResult = testResult
    if testResult < minResult:
        minResult = testResult
    testResult = int(input("Please enter test result (-1 to
finish): "))

print("\nMaximum test result =", maxResult)
print("Minimum test result =", minResult)
```

Formatting output

The data printed above would look much better printed neatly in columns, with column headings at the top.

Format operators

To do this, Python provides **format operators** which are used to produce **formatted strings**.

The **%** operator is a string operator called the **format operator**. Using the format operator, instead of writing something like

```
print(city, temperatureC, localTime)
```

the statement is written as

```
print("%s,%d,%s" %(city, temperatureC, localTime))
```

We can try this out in IDLE to see what the output looks like:

```
>>> city = "London"
>>> temperatureC = 7
>>> localTime = "1200"
>>> print("%s,%d,%s" %(city, temperatureC, localTime))
London,7,1200
>>> print("The temperature in %s was %d degrees C at %s"
        %(city,temperatureC,localTime))
The temperature in London was 7 degrees C at 1200
```

The formatting expression is divided into two parts.

The first part, "%s,%d,%s", contains one or more format specifications. A conversion character tells the format operator what type of value is to be inserted into that position in the string; %s indicates a string, %d indicates an integer.

The second part, %(city, temperatureC, localTime), specifies the values that are to be printed.

Q3

Write statements to do the following:

```
set a = 3, b = 4, c = a + b, d = a * b.
```

Use format operators to print the statements:

```
3 + 4 = 7
```

```
The product of 3 and 4 is 12
```

Chapter 10

Databases and SQL

Objectives

- Learn how data is held in a database so that information can be easily added, deleted, amended and retrieved
- Learn some database terms: table, record, field, primary key
- Write SQL statements to create a database table and add, update or delete data in the table
- Write SQL statements to query a database

Flat file databases

A database is a collection of records held in a number of different tables. In this book we will be concerned only with **flat file databases**, which contain just one table.

Within a database table, data is held in rows, with each row holding information about one person or thing. The data about city temperatures that we held in a text file in the last chapter could be held in a database table like this:

city	temperature	localTime
London	7	1200
Accra	30	1200
Baghdad	20	1500
Winnipeg	-12	0600
New York	14	0700
Nairobi	27	1500
Sydney	22	2300

Chapter 13

Developing an application using Tkinter

Objectives

- Design and implement a data entry form for a given application
- Implement actions to be performed when a button is clicked
- Use a message window to give information to the user
- Close the Tk window and continue or end the program

Sample Application 1

This chapter describes how you might set about developing a GUI application using Tkinter. The sample application will display a data entry screen to enable a teacher or administrator to enter a user ID, first name and surname for a student.

Designing the data input window

You should start by hand drawing a rough design for your form, perhaps something like the image below.

A hand-drawn sketch of a GUI window titled "Student Details form". The window contains three input fields stacked vertically, labeled "Username", "First name", and "Surname". Below the input fields are two buttons labeled "Submit" and "Clear".

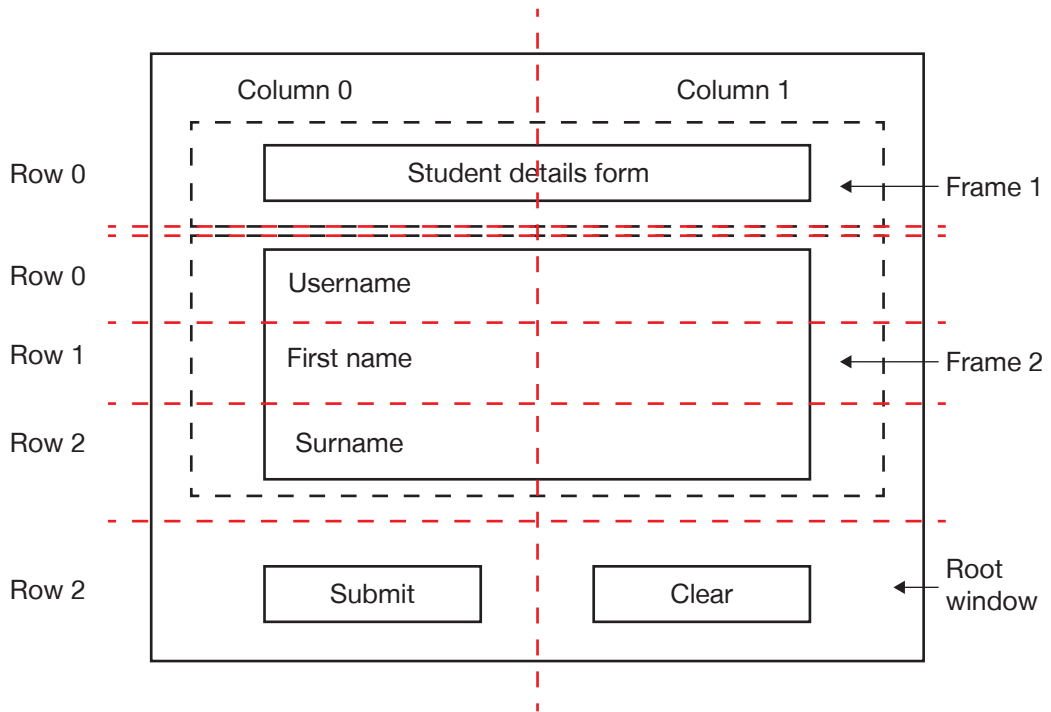
Creating the root window

The parameters of the root window will be set in the same way as in Example 3 of the previous chapter.

```
#create a fixed size window
root = Tk()
root.geometry("270x240")
root.title("Student details")
root.resizable (False, False)
root.configure (background = "Light blue")
```

Creating the frames

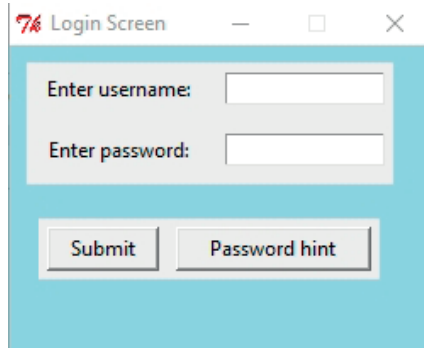
It's useful to draw a grid over the window design so we can easily see in which row and column within a particular frame each widget is to be placed.



Within each frame, the first row is Row 0, the second row is Row 1 and so on. The two buttons are in the main window, not in a frame, so they are in Row 2 with reference to the window, since Frame 1 is in Row 0 of the window and Frame 2 is in Row 1 of the window. Note that the rows do not have to be the same height – their height is determined by their contents.

Sample Application 2

In this application, the user will log on by typing a username and password. If the password is incorrect, an error message will be displayed, the username field and the password will be cleared and the user can press a **Password hint** button to help them get their password correct. In this example, the password is “aaaaaa”. The user name is not checked. The input screen looks like this:



Once the password has been entered correctly, another message window is displayed inviting the user to continue. Once they press **OK**, the message box and the main window close and control passes back to the program, where the user could enter some data, play a game, take a test or perform any other task. In this example the program simply prints “carry on now...” and ends.

The password typed by the user will be replaced by asterisks on the screen, as a security measure. This is achieved by including the parameter `show = "*" in the Entry widget:`

```
entry_password = Entry(frame_entry,width=15,bg="white",show = "*")
```

Using a message box

The messagebox widget is not one of Tkinter’s standard widgets, so you need to include the statement `from tkinter import messagebox` at the top of the program.

A message box is useful for alerting the user to an error or to give them information. In this application we will use two message boxes. The first one pops up with a message when they press a **Password hint** button if they have forgotten their password. It is common practice in many login routines to include a button to click if you have forgotten your password – normally the system will reset the password and email you a new one.



The Python code to display the message box is

```
messagebox.showinfo(title = "Password hint",
                    message = "Hint: Try password aaaaaa")
```

This generates the pop-up window, and the user must click OK to continue.

Once the user presses the **Submit** button, the program checks the password and if correct, displays a message “Password accepted” and a message box to allow the user to continue.

Note: The password and hint in this example are clearly used for testing purposes only; in a real situation, the password and the hint would be something that the user had originally supplied, such as a hint “Grandmother’s birthday” to accompany a password such as GMa221155, to remind them what password they had specified.

Closing the Tkinter GUI

The window named `root` is closed in the `submit` function with the statement

```
root.destroy()
```

Control then passes back to the statement following the `root.mainloop()` statement and the program continues.

The complete listing is shown below.

```
#Program name: Ch 13 Sample app 2 validate password aaaaaa.py
#Program asks user to login, then checks password
#In this program, password is "aaaaaa"

from tkinter import *
from tkinter import messagebox

def submit():
    password = entry_password.get()
    username = entry_username.get()
    messageAlert = Label(root,width = 30)
    messageAlert.grid(row=3, column=0, columnspan=2, padx=5,
                      pady=5)
```

```

if password != "aaaaaa":
    messageAlert.config(text = "Password incorrect")
    entry_username.delete(0,"END")
    entry_password.delete(0,"END")
    entry_username.focus_set()

else:
    messageAlert.config(text = "Password accepted")
    print("password accepted")
    print("Username: ", username)
    print("Password: ", password)
    messagebox.showinfo(title = "Password OK",
        message = "Press OK to continue")
    root.destroy()

# display a message box with a hint for password
def hint():
    messagebox.showinfo(title = "Password hint",
        message = "Hint: Try password aaaaaa")

#create the main window
root = Tk()
root.geometry("250x180")
root.title("Login Screen")
root.resizable (False, False)
root.configure(background = "Light blue")

#place a frame round labels and user entries
frame_entry = Frame(root)
#frame_entry.pack(padx = 10, pady = 10)
frame_entry.grid(row=0, column=0, columnspan = 2,
    padx = 10, pady = 10)

#place a frame around the buttons
frame_buttons = Frame(root)
frame_buttons.grid(row = 2, column = 0, columnspan = 3,
    padx = 10, pady = 10)

#place the labels and text entry fields
Label(frame_entry, text = "Enter username: ").grid(row = 0,
    column = 0, padx = 5, pady = 5)

```

```
entry_username = Entry(frame_entry, width = 15, bg = "white")
entry_username.grid(row = 0, column = 1, padx = 5, pady = 5)

Label(frame_entry, text = "Enter password: ").grid(row = 1,
    column = 0, padx = 10, pady = 10)

# The parameter show = "*" will cause asterisks to appear
# instead of the characters typed by the user
entry_password = Entry(frame_entry, width=15, bg = "white",
    show = "*")
entry_password.grid(row = 1, column = 1, padx = 5, pady = 5)

#place the submit button
submit_button = Button(frame_buttons, text = "Submit",
    width = 8, command = submit)
submit_button.grid(row = 0, column = 0, padx = 5, pady = 5)

#place the Hint button
hint_button = Button(frame_buttons, text = "Password hint",
    width = 15, command = hint)
hint_button.grid(row = 0, column = 1, padx = 5, pady = 5)

#run mainloop
root.mainloop()
print("carry on now...")
```

Sample Application 3

This application allows a user (for example, a teacher) to create a multiple choice test consisting of several questions which could be saved in a text file or database. The input window will look like this:

The skeleton code is given below. There are several new features covered in the code. Notes below the code explain the lines indicated by *Note 1*, *Note 2* etc.

```
#Program name: Ch13 Sample app 3 multiple choice test entry.py
#Program to allow entry of questions for a multiple choice test
#Questions and correct answer are printed
#and could be sent to a file for permanent storage
from tkinter import *
from tkinter import ttk
```

Note 1

```
# functions executed when a button is pressed
def submit():
    if questionNumber.get() == "1":
        print("Test name", testname.get())
        print("Question number: ", questionNumber.get())
        print("Question stem: ", questionStem.get(1.0, END))
        print("Possible answers: ")
        print(possibleAnswers.get(1.0, END))
        print("Correct answer: ", correctAnswer.get(), "\n")
```

Note 2

Index

A

arithmetic operators, 3
ASCII code, 38
assignment statement, 5
attribute, 72
augmented assignment
operators, 6

B

Boolean
conditions, 18
expressions, 19
variable, 18
break statement, 26

C

clear() function, 100
close() method, 77
combo box, 108
comma separator, 6
comments, 12
commit() method, 77, 85
compiler, 1
concatenate, 2
connection object, 76
console window, 12
container data types, 44
conversion functions, 15
cursor object, 77

D

data type
Boolean, 4
numeric, 2
string, 2
database
flat file, 71
query, 72, 81
database table
add new record, 73
create, 77
delete a record, 74
update a record, 74
del, 46

dictionary, 44
methods, 45
div, 3
docstring, 52

E

Editor window, 11
elif, 18
end
in print statement, 8, 63
escape sequence, 7, 32
executemany, 80

F

field, 60, 72
file
open, 60
process, 65
read, 61, 63
write, 64
flag, 40
flat file database, 71
float() function, 15
floating point, 2
for loop, 22
format
check, 37, 41
modifier, 69
operator, 68
output, 68
frame, 96
place in window, 99
function, 16, 50, 55
built-in, 50
conversion, 15
parameters, 52
programmer-written, 51
return values, 52

G

Geometry Manager
Grid, 91
Pack, 91
Place, 92
global variable, 57

Graphical User Interface, 88
GUI, 88

I

identifier, 4, 72
IDLE, 1
indexing strings, 26
input statement, 9
inputting numbers, 14
int() function, 15
integer, 2
integer division, 3
integrated development
environment, 1
interactive
mode, 1
window, 2
interpreter, 1
interrupt execution, 28, 115
iteration, 17

J

justify
left or right, 109

K

kill program, 28

L

lambda keyword, 48
length check, 37
list, 29
append, 31
methods, 30
two-dimensional, 33
local variable, 56
logic error
finding, 115
logical operator, 4
long statement, 8

M

mainloop(), 93, 109
message box, 102



method, 16
 methods
 string, 12
 mod, 3
 mode
 append, 64
 write, 64
 multi-line statement, 8
 mutable, 44

N

nested loops, 23
 nested selection statement, 19
 newline, 7
 None keyword, 30

O

object, 4, 12
 open
 mode, 61
 operator
 logical, 4
 relational, 4
 output
 formatted, 67

P

pack() method, 90
 padding, 93
 parameters, 52
 pdb module, 115
 placing widgets, 91
 planning a program, 110
 primary key, 72
 print, 6
 on same line, 8
 procedure, 50, 55
 pseudocode, 111
 Python Shell window, 2, 11

Q

quote mark, 2

R

randint() function, 20
 random number, 20
 record, 60, 72
 regular expression, 41
 relational operator, 4
 return values, 52, 54, 55
 root window, 57, 98
 round() function, 3

S

script mode, 1, 11
 selection, 17
 separator, 7
 sequence, 17
 Shell window, 2
 slicing strings, 27
 sort
 list, 47
 table, 48
 spaces
 in statements, 4
 SQL, 72
 command, 77
 SQLite, 76
 str() function, 7, 15
 string, 2
 string processing, 25
 Structured Query Language, 72
 submit() function, 100
 subroutine, 50
 syntax error, 13

T

tab, 7, 32
 terminate program, 28
 test plan, 113
 text file, 59
 Tkinter module, 88
 trace, 115
 trapping errors, 38, 84
 triple quotes, 8, 52
 try ... except, 38, 84
 ttk module, 109
 tuple, 35
 type check, 38

V

validation, 37
 variable
 global, 57
 local, 56
 name, 4, 5

W

while loop, 24
 widget, 90
 button, 90
 combo box, 108
 label, 91
 place in window, 91
 text, 109
 window
 close, 103
 master, 98
 setting parameters, 93
 with connection, 85

Learning to Program in Python >>



This book is intended for individuals and students learning to program. You may already have done some programming in other languages, but not be familiar with Python. Novice programmers should work through the book sequentially, starting at Chapter 1. It will also be a useful reference book for students on a programming course or anyone working on a programming project.

It teaches basic syntax and programming techniques, and introduces three built-in Python modules:

- **Tkinter**, used for building a graphical user interface, which is an option that some users may like to include in their project work.
- **SQLite**, which enables the creation and processing of a database from within a Python program. This provides an alternative to writing to a text file when data needs to be stored and retrieved.
- **pdb**, Python's debugging module, which can be used to help find elusive logic errors.

Questions and exercises are included in every chapter. Answers to these, as well as over 120 Python programs for all the examples and exercises given in the book, may be downloaded from www.pgonline.co.uk. These programs enable users of the book to try out the in-text examples and check possible solutions to the exercises.

Cover picture:

'Fosse No.4'

Oil on linen,

30x30cm © Barbara Burns 2015

www.slaneart.com



PG ONLINE

ISBN: 978-1-910523-11-7



9 781910 523117