Coding with Python

Bill Black

September 17, 2025

How to run Python Programs

Python is an *interpreted* language, and the Python interpreter must be installed on a computer in order to run Python programs. As the method of installing Python differs for each computing platform (e.g. Windows, Linux, OS X, Raspberry Pi, iPads and other tablets) we won't cover that in this short introduction.

Instead, we'll use a Web browser to run Python programs on a server where it has already been installed. We're going to start by following an on-line tutorial at **https://www.w3schools.com/python/**.

While following the tutorial, after explaining a feature of Python, the tutorial provides links labelled "Try for yourself". Each time you follow such a link, a split window appears, with a pre-written Python program on the left, and an output pane to the right. If you press the "Run" icon, the program's output appears in the right-hand pane. You can reinforce your understanding of how the program works by making a small change, first to its input values, then to the program statements, and re-running the program.

To develop programs independently of where you are in the tutorial, you can practice at **https://www.online-python.com/** but that is a limited environment that does not support file handling. If you want to be able to run the example programs for yourself, and to develop projects of your own, I recommend registering for a free account at **https://www.pythonanywhere.com/**.

Registering with PythonAnywhere

At **https://www.pythonanywhere.com/**, the sign-up page currently looks like Figure 1. Press the green lozenge to get started, and on the next screen, press the blue button labelled "Create a beginner account". You then have to provide a username, your email

address, and a password. Your email will be validated as part of the sign-up process, and when you log in, you will see your account Dashboard, as in Figure 2.

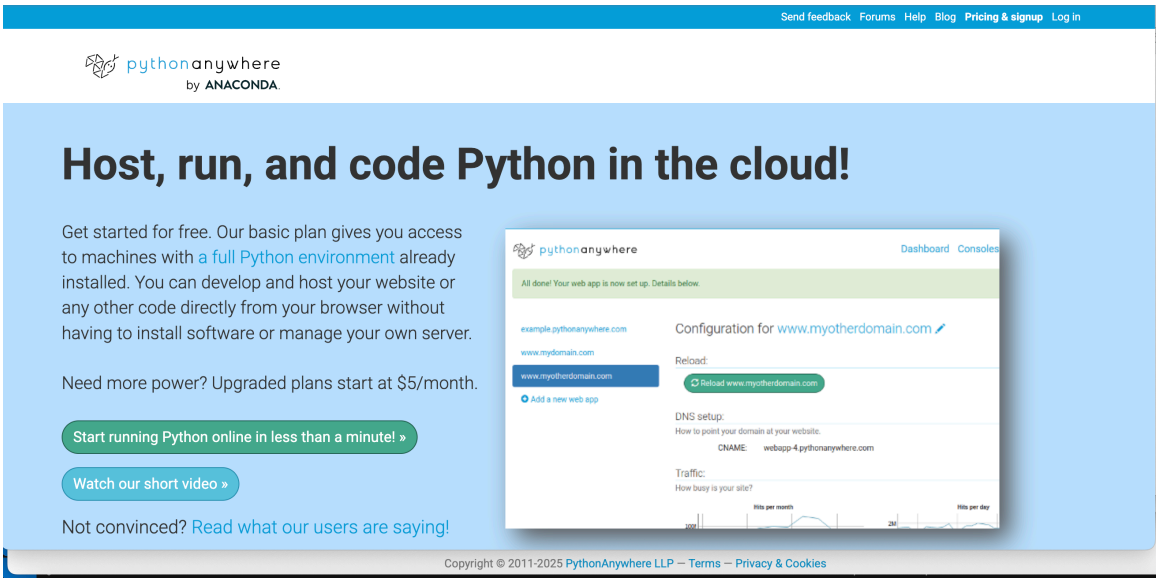

Figure 1: The sign-up page at PythonAnywhere

The PythonAnywhere Dashboard

From the Dashboard, you can work with Files, where you create programs (or upload them from your device), and you can work with Consoles, where you run programs. The other menu options are beyond the scope of this tutorial.

Working with files in PythonAnywhere

The File Manager page provides the basic facilities of the equivalent on your desktop computer. On the left, you can view, create and delete directories (folders), and to the right, you can list the files in a directory, create a new file, delete existing ones, or click on the file name to open it in an editing pane.

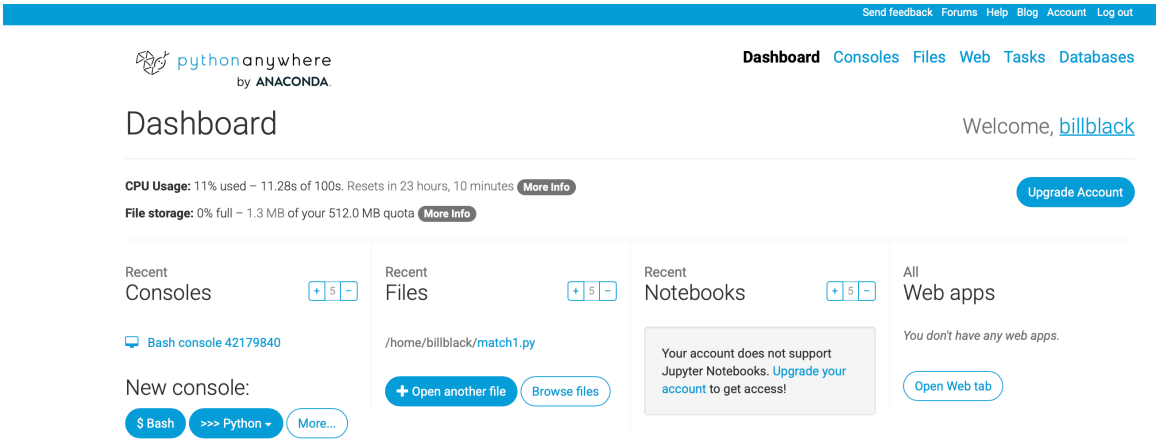

Figure 2: The PythonAnywhere Dashboard

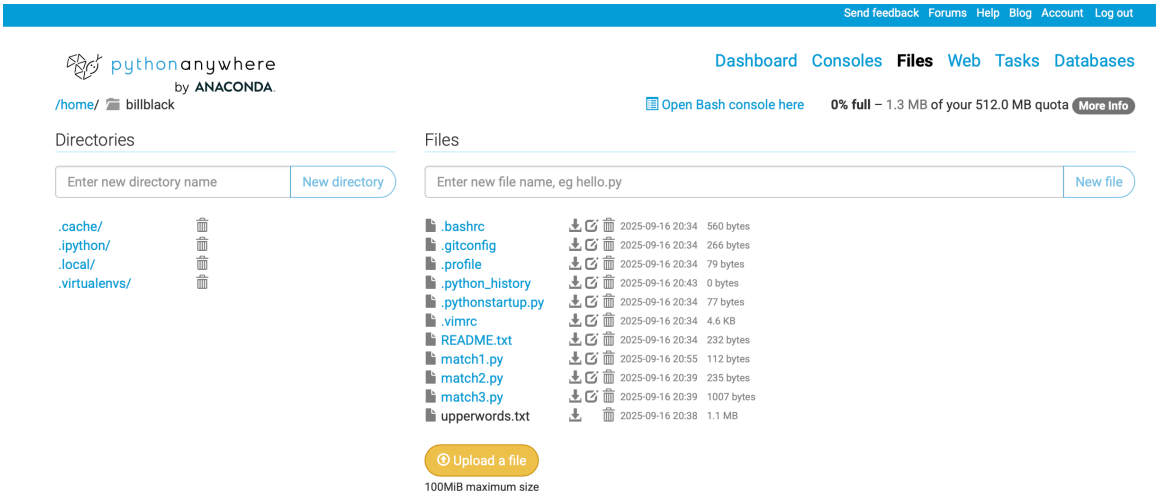

Figure 3: The PythonAnywhere File Manager

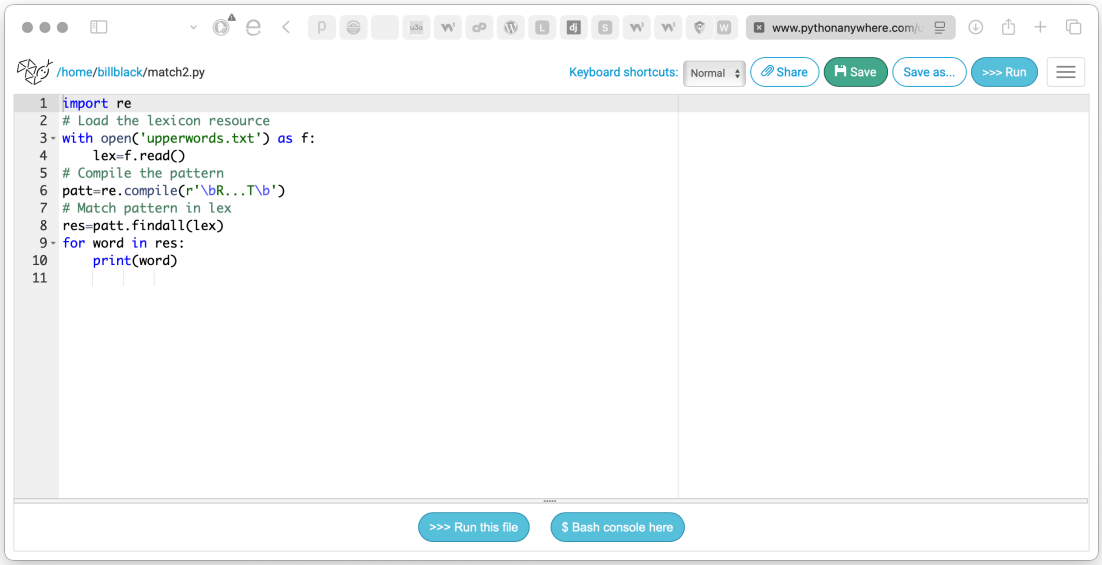

Figure 4: A program file in the editing window.

Editing and Running a program in PythonAnywhere

Figure 4 shows one of the example programs from the notes below in the editing pane. You can edit the program code in this pane, using the keyboard and mouse. When you are ready to run the program, there are two buttons that will do that for you. If you press either, a console appears below the editing pane, with white typewriter-style output on a black background. There, you see the output of the program.

Getting the sample programs

The sample programs for the Example Project discussed next are available online at **http://wjblack.co.uk/python-resources**. If you have set up a PythonAnywhere account, or obtained a Python environment otherwise, you should download the files there to a folder on your own computer, and then upload them to your PythonAnywhere File Manager.

Example Project

For the example, we will illustrate several capabilities of Python, notably text processing and file handling, all within a program that is quite short in length.

Goal Write a program that will help to solve crossword or codeword clues, by finding words that match a pattern with known letters in some positions but 'wildcards' in the other positions.

The program should not attempt to match solutions with crossword clues, cryptic or otherwise, just to solve the constraints of the known letters.

Resource The program will need access to a lexicon or collection of words and phrases. A list of over 100,000 English words and phrases will be provided.

Strategy We'll do the program in stages, first working on the matching mechanism with a small sample of words, then loading the full lexicon and matching against that, and finally making some incremental improvements.

First elaboration of the program

Figure 5 shows a first, simplified, version of the program. The line numbers are not part of the program code, but are there to allow reference to individual lines of code.

Line 1 is needed because the methods for matching strings are in the `re` package, a library of functions implementing matching of *regular expressions*. These are used widely in programming languages and also in advanced search and replace tools in wordprocessors and editors.

```
1. import re
2. patt=re.compile('r...t')
3. res=patt.findall('robot react trump reset')
4. for word in res:
5.     print(word)
```

Figure 5: First version of the program

In Line 2, the pattern `r...t` is an example of a regular expression (RE). In this RE, the character literals `r` and `t` stand for themselves and '.' stands for any single charac-

ter. The whole pattern will match any 5-character string (whether a word or not) that begins 'r' and ends 't'.

In Line 3, the pattern is applied to the string `'robot react trump reset'` and a collection of all the matches is placed in the variable `res`.

Line 4 says to do what follows for every individual value, which we'll call `word`, within the sequence `res`. What follows in line 5 is the instruction to print out the current `word`.

Version 2 loading a dictionary from a file

Figure 6 shows an elaborated version of the program. We are dispensing with the line numbers, and instead adding comments within the program to explain the code that follows. These are the lines that start with #.

```
import re
# Load the lexicon resource
with open('upperwords.txt') as f:
    lex=f.read()
# Compile the pattern
patt=re.compile(r'\bR...T\b')
# Match pattern in lex
res=patt.findall(lex)
for word in res:
    print(word)
```

Figure 6: Second version with pattern matched against an entire dictionary

Version 3 inputting the pattern to match from the user

Version 2 is not very flexible, as the pattern to search for is a string literal within the program code.

To make the program really usable, we have to be able to use it with whatever answer pattern its user wishes to use. In turn, the user needs to be told how to input a pattern. These requirements are dealt with in the version listed in Figure 7.

Python offers a convenient way to write multi-line string literals, which we use in the assignment to `banner`.

The input function outputs a prompt and then suspends the program's operation until a response has been typed.

```
import re
########################################
# Version 3 - inputs pattern from user   #
########################################
# Load the lexicon resource as a string
with open('upperwords.txt') as f:
    lex=f.read()
# Prompt for the pattern to match
banner = '''

u3aTod Crossword Helper

Helps you solve a crossword clue or a word in a codeword,
When prompted, enter a pattern matching a partially-known
answer. Type the letters you know, and write a dot in
place of the letters you don't know.  Example: ..T.O. which
would match PYTHON. Max length 13 characters.

'''
print(banner)
user_pattern=input('Enter your crossword answer pattern: ')
user_pattern=r'{}'.format("\\b"+user_pattern+"\\b")
# Compile the pattern
patt=re.compile(user_pattern)
# Match pattern in lex
res=patt.findall(lex)
for word in res:
    print(word)
```

Figure 7: Third version with pattern input from user

As in version 2, we have to add `\b` (word boundary markers) to the beginning and end of the pattern to limit the search to whole words. The statement

```
user_pattern=r'{}'.format("\\b"+user_pattern+"\\b")
```

does this for whatever pattern the user has supplied. The escape character `\` has to be preceded by another `\` in a string literal. The `r` at the beginning of the pattern forces the following literal to be treated as a 'raw string', otherwise the pattern matching will not succeed.

Version 4 improving the layout of the output

There are numerous ways to improve on the operation of the program, all of which are perfectly possible in Python. Here are some examples:

- Allow the user to use alternative wildcards, e.g. ? as used in the online program OneAcross.
- Allow the pattern to be written in lower-case or mixed-case letters, and convert these to the equivalent upper-case letters to match the lexicon, which has been converted to all upper-case.
- Validate the user's input to ensure that only letters and wildcards are used, since crossword answers don't contain numbers or other non-alphabetic characters. Also that the pattern is not too long for the maximum-length crossword answer of 13 letters.
- Have the program repeatedly prompt, input and process a pattern rather than do it once and then stop.
- Run the program on the web.
- Format the output so that when there are a large number of possible matches, they are laid out in columns, several words to a line, minimising the need for scrolling.

Version 4 of the program will tackle the second and the last of these enhancements, and we'll set some of the others as exercises. This will be a convenient illustration of Python's *conditional* and *iterative* statements.

Let's clarify what we want to do. A simple solution might be to output up to five solutions to a line. Since patterns and the words they match can vary in length from 3

to 13 letters, we'd be better making sure that the length of each line of output is what determines when to move to the next line rather than the number of words.

What we'll need to do, while there is more output available, is to add words to a string, which we'll call buffer. When the length of the string exceeds our chosen threshold, we print out the buffer and then empty it ready for the next output word. Before doing anything, we *initialize* the buffer to the empty string.

The last two lines in this version are expanded as follows:

```
buffer=''
for word in res:
    if(len(buffer))>60:
        print(buffer)
        buffer=''
    buffer+=word
    buffer+=' '
print(buffer)
```

With the earlier line that creates the pattern to search for modified to convert lowercase to upper, as follows:

```
user_pattern=r'{}'.format("\\b"+user_pattern.upper()+"\\b")
```

the program output is as in Figure 8

```
u3aTod Crossword Helper

Helps you solve a crossword clue or a word in a codeword,
When prompted, enter a pattern matching a partially-known
answer. Type the letters you know, and write a dot in
place of the letters you don't know.  Example: ..T.O. which
would match PYTHON. Max length 13 characters.


Enter your crossword answer pattern: r...t
RABAT RASHT REACT REBUT REFIT REMIT REPOT RESET RESHT REVET RIANT
RIGHT RIVET ROAST ROBOT ROGET ROOST
(.venv) williamblack@WILLIAMs-MacBook-Air py %
```

Figure 8: Output of Version 4 of the program

Exercises

1. Write a Python statement that will convert all letters within the contents of a string variable called `text_in` to upper-case, and assign the result to `text_out`. **HINT** Look in the w3schools Python tutorial under the heading **Strings**, and seek out the list of string functions.
2. Similarly, write a statement that will convert any question marks in a string variable to full stops, assigning the result to another string variable.
3. Write a function called `standardize` which will have a parameter called `txt`, and return a string normalised to upper-case and with all question marks replaced by dots.
4. Amend your solutions to the last two problems so that any non-alphabetic character, not just a ?, is changed to a dot. **HINT** You need to look in the documentation for a function that replaces regular expressions and not literal strings.
5. The *Guardian* puzzle *Wordiply* gives players a short character sequence, which need not be a whole word, and challenges them to find the longest English words that contain that as a substring. The task here is to convert the crossword helper program to solve this challenge, starting with version 1. The key to the solution is to be found in regular expressions. Whereas the crossword helper uses single character wildcards, this problem can be solved with a regular expression using wildcards that represent sequences of zero or more characters.