Introduction to Computer Programming

Course Notes

Contents

[Introduction to Computer Programming 1](#_Toc514957171)

[1. Introduction 4](#_Toc514957172)

[1.1 The Course 4](#_Toc514957173)

[1.2 The Instructor 4](#_Toc514957174)

[1.3 Requirements 5](#_Toc514957175)

[2. What is computer programming? 5](#_Toc514957176)

[2.1 Computer is a VERY Obedient Servant 5](#_Toc514957177)

[2.2 Similar activities 6](#_Toc514957178)

[2.3 What’s Easy about Computer Programming 7](#_Toc514957179)

[2.4 What’s Not Easy about Computer Programming 7](#_Toc514957180)

[2.5 Do I care Even if I don’t Actually Program? 8](#_Toc514957181)

[3. Let’s take a look! 8](#_Toc514957182)

[4. Adding a few things 9](#_Toc514957183)

[4.1 Variables – Places to store data/values 9](#_Toc514957184)

[4.2 Printing – send stuff to the screen 10](#_Toc514957185)

[4.2.1 Exercise: 10](#_Toc514957186)

[4.3 Text – We speak, usually in words 11](#_Toc514957187)

[4.3.1 Text strings – 11](#_Toc514957188)

[4.4 Decisions – Without them choices are few 12](#_Toc514957189)

[4.4.1 Exercise: Sum the numbers 14](#_Toc514957190)

[4.5 Input – What’s the user want 14](#_Toc514957191)

[4.5.1 Exercise: Let the user tell 15](#_Toc514957192)

[4.6 Comments – what we want 15](#_Toc514957193)

[5. Computers / Programming an Overhead Look 15](#_Toc514957194)

[5.1 The Computer – Does the program’s work 15](#_Toc514957195)

[5.1.1 Output – what we get: see/hear/feel 15](#_Toc514957196)

[5.1.2 Input – what we give: tell/ask 16](#_Toc514957197)

[5.1.3 CPU (Central Processing Unit) – the actor 17](#_Toc514957198)

[5.2 Programming Language 17](#_Toc514957199)

[5.3 Output 18](#_Toc514957200)

[5.4 Input 18](#_Toc514957201)

[5.5 Computation - Arithmetic 18](#_Toc514957202)

[5.6 Distinction between integer (whole number) and floating point (fractions) 19](#_Toc514957203)

[5.7 “=” 🡪 data on right travels to place on left 19](#_Toc514957204)

[5.8 Encapsulation Part 1 – Functions 19](#_Toc514957205)

[5.8.1 Exercise – special product 20](#_Toc514957206)

[5.8.2 Exercise – staircase function 20](#_Toc514957207)

[5.9 Encapsulation Part 2 – Classes 21](#_Toc514957208)

[5.10 Compare 21](#_Toc514957209)

[6. A Computer Programming Roadmap 21](#_Toc514957210)

[6.1 Problem – What’s bothering you, your boss, … 21](#_Toc514957211)

[6.1.1 Goal(s) – What do we want to do about it 22](#_Toc514957212)

[6.1.2 Plan(s) – Exactly, How do we want to do this? 22](#_Toc514957213)

[6.1.3 Create 22](#_Toc514957214)

[6.1.4 Test – Let’s see if it does what we wanted. 22](#_Toc514957215)

[6.1.5 Distribute - giving it out 23](#_Toc514957216)

[6.1.6 Celebrate – 23](#_Toc514957217)

[6.1.7 Do it again – Better? On to the next? 23](#_Toc514957218)

[6.2 Computer Programming Languages 23](#_Toc514957219)

[6.2.1 Python – 23](#_Toc514957220)

[6.2.2 Java 23](#_Toc514957221)

[6.2.3 C / C++ 23](#_Toc514957222)

[6.2.4 Assembly Language – Closest to the computer hardware 23](#_Toc514957223)

[7. Let’s Write a Program 24](#_Toc514957224)

[7.1 Goals – What’s it going to do? Or at least, what would be nice? 24](#_Toc514957225)

[7.2 Elaborate the goals of our program 24](#_Toc514957226)

[7.3 Prioritize – e.g. No 3D graphics display first round 24](#_Toc514957227)

[7.4 The Plan – How are we going to make it do what we want? 24](#_Toc514957228)

[7.5 Or at least, some of what we want? 24](#_Toc514957229)

[7.6 Displaying Results/Request 24](#_Toc514957230)

[8. I’m thinking of a number… 24](#_Toc514957231)

[8.1 Game Description 25](#_Toc514957232)

[8.1.1 Traditional “20 questions” 25](#_Toc514957233)

[8.1.2 Program picks number and player guesses: “Is it greater than 5” continually. 25](#_Toc514957234)

[8.1.3 Main player choices are “> N”, “<N”, “=N” where N is a number, and “Q” for quit 25](#_Toc514957235)

[8.2 Plan – Iterate 25](#_Toc514957236)

[9. Appendix 25](#_Toc514957237)

[9.1 Online Meeting Setup 25](#_Toc514957238)

[9.2 Using PythonAnywhere 26](#_Toc514957239)

[9.2.1 Connecting to PythonAnywhere 26](#_Toc514957240)

[9.2.2 Reconnecting to PythonAnywhere Account 38](#_Toc514957241)

[9.3 Online Python Resources 39](#_Toc514957242)

[9.3.1 PythonAnywhere 39](#_Toc514957243)

[9.3.2 python.org 39](#_Toc514957244)

[9.3.3 Other Online Python Resources 39](#_Toc514957245)

[9.3.4 Learning Python 41](#_Toc514957246)

# Introduction

## The Course

This is a short course for those who have little or no experience in computer programming. The students will gain insight into programming ideas and goals. We will be using Python, a very popular and powerful programming language. My aim is to help the participants decide if further programming study might be for them. I will introduce some of the central ideas, techniques, and styles I’ve gained over my career. Our method will be to get the students to do some actual programming. This is where the actual “feel” of programming can be experienced. Participants must bring their laptop computers and have basic computer skills such as powering on, creating and modifying text files, and browsing the Internet.

## The Instructor

Ray (Charles **Ray**mond Smith) - [raysmith619@alum.mit.edu](mailto:raysmith619@alum.mit.edu)

* Years of Programming – engineering, scientific, financial
* Programming environments – embedded, systems, application
* Languages – C, C++, Perl, Java, Python, Assembly, shell, Fortran, PL1, …
* Currently investigating computer graphics/game programming for the very young (3-8 years)

## Requirements

* Interest in programming
* Laptop computer

# What is computer programming?

* Controlling the computer
* Telling the story that gets done
* The computer language IS the computer
* Providing the instructions to achieve the goal
* Making the story reusable
* Like physical construction but “magical”

## Computer is a VERY Obedient Servant

Does exactly what you tell it – hopefully what you want

Example: Car

* Don’t need to know what’s under the hood
* Most cars are alike
* Standard parts
  + Inputs/controls
    - Steering wheel, Pedals, Signals, …
  + Outputs / Signals
    - Wheels, Lights, Windows
  + Defined actions
  + Same action every time – almost – e.g. forward / backward
* Car gets the blame…

## Similar activities

* Abstract / Indirect

Cooking recipes

Travel directions

Assembly instructions

* Physical – What you see is what you get

Construction - Roads, Buildings

* Like Computer Programming

Build once – use many times

Builder and user(s) often different

Building errors show up many times

Building errors don’t always show up immediately

* Not so like Computer Programming

Physical materials

* Hard to copy
* Hard to manipulate
* Easy to see

## What’s Easy about Computer Programming

* No so physical

Creation / copying / moving is a snap

Things don’t wear out – works today, tomorrow, forever

Easy to change

Working on new one, while the old one is running

## What’s Not Easy about Computer Programming

* No so physical

Can’t “see” it

Everything is connected – unexpected consequences

Often not easy to tell how close to done

## Do I care Even if I don’t Actually Program?

* We depend on programs, even if written by others
* Better knowledge of what is possible / difficult / easy
* To more easily deal with, instruct, cope with results.
* Understand alternatives – e.g. get someone else to do it

# Let’s take a look!

* Seeing is believing
* Lots left unsaid
* Try, Try, Try
* Think arithmetic / algebra
* Python as a super calculator

Using online Python: <https://www.pythonanywhere.com>

Please refer to the Appendix for detailed instructions on how to obtain access to PythonAnywhere.

Note that the “**>>>** ” and “**…** ” are prompts displayed by the Python interpreter as guides and **NOT** typed by you.

Python 3.6.0 (default, Jan 13 2017, 00:00:00)

[GCC 4.8.4] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>> 2+2

4

>>> a=2

>>> b=3

>>> a+b

5

>>>

Exercises:

1. Assuming: salary: 1000, New salary: 1250
2. What’s the raise?
3. What’s the percent raise?
4. If the new salary was 1500, 1600, 1700?
5. How many seconds :
6. In a day?
7. In a week?
8. in a year of 365 days?
9. In a week?
10. Between 8AM Monday and 5PM Friday?

Using online Python: <https://www.pythonanywhere.com>

Enter the following in the top (code) window

then click the **Run** button

a = 4

b = 3

print(a+b)

The following should be displayed in the lower (output) window

5

* How many ways can you make interesting calculations?

# Adding a few things

## Variables – Places to store data/values

* Facilitates using our data/results
* Data stores have names
* Names are of the form:

Start with a letter (a-zA-Z\_)

Followed by zero or more digits (0-9) or letters

examples: a, a1, my\_variable, a2or3

NOT: 1, 1a, my-variable, “ray,smith”

* Storing a value into a variable:

variable\_name = value

Examples:

a = 1

a1 = 5.7

my\_variable = my\_variable + 1

## Printing – send stuff to the screen

* **print(***value***)** - *value* goes to the screen
* **print(***value1, value2, …***)** - *values* go to the screen
* End of line added after value(s)
* Example:

Using online Python: <https://www.pythonanywhere.com>

print("Hello World - all on the same line")

print("""Multi-line string

on multiple lines.

Another line goes here.

""")

Output:

Hello World - all on the same line

Multi-line string

on multiple lines.

Another line goes here.

* NOTE: Older Python versions(2.x) used print “……” without parentheses

### Exercise:

1. Calculate 5 years total interest on $1M, given an annual interest of two percent.

Name the variables

What’s the calculation

Extra Credit: 10 years ?, 100 years?

Hint (for lesser ammounts):

# $100, 3 years, %1 interest, very short variable names

p = 100

i = .01

mult = 1 + i # Multiplier == 1 + interest

tot = p \* mult \* mult \* mult # Three years

ti = tot - p

print("Total interest for 3 years:", ti)

Output

Total interest for 3 years: 3.0301000000000045

## Text – We speak, usually in words😊

### Text strings –

“…..stuff….” placed all on one line

OR

“””…stuff may be multiple lines “”” (three adjacent “)

Example:

Using online Python: <https://www.pythonanywhere.com>

prompt = "Please Enter your answer:"

print("Prompt:", prompt)

help\_message = """

We are glad you are participating

Your choices are as follows

1. Go home

2. Stay here

3. Speak up

"""

print("help\_message:", help\_message)

Output:

Prompt: Please Enter your answer:

help\_message:

We are glad you are participating

Your choices are as follows

1. Go home

2. Stay here

3. Speak up

## Decisions – Without them choices are few

The primary Python testing / decision constructs are:

**if** – Either it is or it isn’t

if is true **else:** false

**while** – Loop Till not True

**for** – Loop from beginning value till ending value

Decision code follows the form:

*Testing-construct boolean-value-expression* **:**

*Indented-code-lines*

Boolean-value-expression examples:

*Value* compare-operator *Value*

Where compare-operator can be:

**>**, **<**, **==**, **!=**, **>=**, **<=**

*Indented-code-lines* – code lines indented to the same amount

* One or more lines
* Can include additional decision constructs
* Special Cases:

**break** – quit the loop

**continue** – stop doing the indented portion and restart at the beginning of this portion

**pass** – do nothing – just a place holder

* Often used, when there is nothing to do here, but the language rules says we have to have something here

Example:

Using online Python: <https://www.pythonanywhere.com>

if 2 < 3:

print("if true=>", "2<3", 2<3)

if 2 > 3:

print("if true=>", "2>3", 2>3)

print("After if 2 > 3")

if 2 > 3:

print("if true=>", "2>3", 2>3)

else:

print("else false=>", "2>3", 2>3)

Output:

if true=> 2<3 True

After if > 3

else false=> 2>3 False

Example:

n = 1

max = 3

while n <= max:

print("n:", n, "max:", max)

n = n+1

print("After loop")

Output:

n: 1 max: 3

n: 2 max: 3

n: 3 max: 3

After loop

Example:

print("for loop")

for i in range(1,3):

print("i:", i)

Output:

for loop

i: 1

i: 2

### Exercise: Sum the numbers

1. Sum the numbers from 1 to 100
2. List the numbers from 1 to 50, that are evenly divisible by 2, 3, 4, and 5.

Hint: I found the following by an Internet search for

python evenly divisible

n % k == 0

evaluates true if and only if n is an exact multiple of k. In elementary maths this is known as the remainder from a division.

Example:

The Following code:

if 10 % 2:

Print(“10 % 2”

if 10 % 3:

Print(“10%3”)

Outputs:

10 % 2

1. Sum the numbers whose sum is less than 1000000, listing the sum and the last number added.

## Input – What’s the user want

With input we can reuse the program for different situations without making changes to the program.

* **input()** – returns line user typed on keyboard
* **input(**“Please enter your answer”**)** – prompts user, then returns text entered by the user (usually on the keyboard)

### Exercise: Let the user tell

1. Tell the users you are going to sum numbers
2. Ask the user to enter the sum limit (Limit).
3. Sum the numbers whose sum is less than Limit, listing the sum and the last number added.

## Comments – what we want

* What we want, think, expect
* Not “code” but more important
* Comment 🡺 what you want vs. code 🡺 what you did
* **#** *text to indicate what you want, think, …end of line*
* **“””** *Any text you choose, really used as a code text string, but often used as a programming comment. This comment often goes on for several lines ending at the next* **“””**

# Computers / Programming an Overhead Look

## The Computer – Does the program’s work

### Output – what we get: see/hear/feel

#### Display

* Dynamic
* Multi-dimensional
* Complex

#### Printer

* Time honored
* Simple

### Input – what we give: tell/ask

#### Keyboard

* Simple
* Easy to use
* Easy to program

#### Mouse / Touch screen

* Two dimensional
* Smart phones 🡪 becoming more accepted / expected

#### Other cool stuff

### CPU (Central Processing Unit) – the actor

#### Calculator – adding, subtracting, …

#### Storage – where we store stuff (data)

##### Memory – Fast

##### Storage – Large (Rotating Memory – Disks)

## Programming Language

* Computer languages are the languages in which we tell the computer what and how to do what we want.
* Our programs are written in these languages.
* The parts of the computer language correspond to instructions to control specific computer parts.
* As computers are evolving, the link between the physical (computer) and logical (programming language) changes.
* As programming techniques evolve the programmer needs to think about the physical parts in less detail.
* Still, it is often good to keep in mind that the computer program execution, at some level, will execute on actual hardware.
* Most often it is productive to concentrate on the programs goals.
* In the early stages (specification, planning), often ignore the physical computer attributes.

## Output

* print – Text to the screen or file
* write – bulk output to file
* Graphics – pictures, etc. – more complicated

## Input

* input() – get a line from user / keyboard
* read – bulk input from file

## Computation - Arithmetic

* Heavy use of arithmetic and algebraic notation
* Slight relaxation in semantics

“=” 🡺 data on right travels to place on left

“<”, “>”, “==” 🡺 compare value on left with that on right

## Distinction between integer (whole number) and floating point (fractions)

## “=” 🡪 data on right travels to place on left

## Encapsulation Part 1 – Functions

* Divide and Conquer
* Parts of the problem placed in functions
  + Called (invoked/started) with *name*(comma-separated list of parameter values
* Function Definition – what it is going to do
  + Name – same style/restrictions as variable name
  + Parameters – data which is passed to the function
    - Comma-separated list of parameter specifications
    - Simplest is a set of variable names
  + Body – what the actions are
    - Regular code
    - Uses data passed in parameters by parameter names
* Example:

def add2(value1, value2):

sum = value1 + value2

return sum

* Function Calling / Invocation
* How do we call the function?
  + Called with the function name
  + Including parenthesized list of values
  + The values are passed to the function
  + Example:

sum1 = add2(1,2) # sum1 gets 1+2

sum2 = add2(3,4) # sum2 gets 3+4

### Exercise – special product

Write a function product(factor1, factor2, factor3) that returns the product of the values factor1, factor2, factor3. Test it on the following: .5, .4, .3; 1, 2, 3; -1, -1, -1;

### Exercise – staircase function

Write a function staircase(first\_n, last\_n) that returns the sum of the numbers first\_n, first\_n+1, …, last\_n. Test it on the following: 1,10; 1,100; 5, 10; 5,5; 10,1;

Did you use:

**for** – why?

**while** – why?

neither – why?

neither – why?

## Encapsulation Part 2 – Classes

* Data and Function
* Facilitate Object Oriented Programming
* For later discussion

## Compare

# A Computer Programming Roadmap

## Problem – What’s bothering you, your boss, …

In most endeavors, the most important thing is to decide what needs “fixing”, creation, …

### Goal(s) – What do we want to do about it

* What part or parts do we want fixed, created, …
* Which things are most important?
* Which need to be done first, last, before others.

### Plan(s) – Exactly, How do we want to do this?

* Has this or part of this, already been done?
* Can/Should we do part of this first?
* Is this a programming task?
* Can we divide this up in to smaller parts?
* What’s the most fun part?

### Create

* More detailed planning - choose program layout

Modules

Subroutines / functions

Variables

Major calculations

* Write the code – text of the program

May include checking reference material, e.g. Googling

Possible experiments about how to do *X*

Visual inspection – can save overall time / effort / pain

### Test – Let’s see if it does what we wanted.

Before we expose our work to others, we probably should examine it.

The bigger the exposure, the more important the examination.

The more complicated - the more testing is useful.

Divide and conquer.

### Distribute - giving it out

* Packaging
* Documentation – user manual, online help

### Celebrate – 😊

### Do it again – Better? On to the next?

## Computer Programming Languages

* Many popular languages in use today
* Most similar in form / function

### Python –

* Highly popular programming language today
* This course will use Python
* Facilitates Object Oriented Programming

### Java

* Facilitates Object Oriented Programming

### C / C++

* Highly efficient
* Often the basis for computationally intense modules / packages
* C++ - Object Oriented version of C

### Assembly Language – Closest to the computer hardware

* Different for every hardware
* Still not the hardware
* Little used today
* Even, when used, usually only when necessary

# How to Write a Program

How should we write a program? How should e write a good program? Does it need to be the best?

## Goals – What’s it going to do? Or at least, what would be nice?

## Elaborate the goals of our program

## Prioritize – e.g. No 3D graphics display first round 😊

## The Plan – How are we going to make it do what we want?

## Or at least, some of what we want?

## Displaying Results/Request

# I’m thinking of a number…

## Game Description

### Traditional “20 questions”

### Program picks number and player guesses: “Is it greater than 5” continually.

### Main player choices are “> N”, “<N”, “=N” where N is a number, and “Q” for quit

## Plan – Iterate

# Appendix

## Online Meeting Setup

1. Click the following or paste the following address in to your browser address:

<https://zoom.us/j/2821636776>

1. If you are new to Zoom:
2. After download completes, click Zoom\_Launcher.exe in lower left corner of screen.
3. If you see “Do you want to run this file? …Zoom\_Launcher…” click the “Run” button.
4. To the prompt: “Please Enter Your Name”, enter your name, then click **Join.**
5. If you are in the same room as the speaker, please mute your microphone by clicking the microphone icon in the lower left corner of the Zoom window.

## Using PythonAnywhere

PythonAnywhere is an online resource which can easily provide quick access to Python. We will be using it extensively for this course. The following steps provide you access.

### Connecting to PythonAnywhere

* Via Web Link

Select the following link (CTL Mouse Right-Click) or paste Web link text in to browser address field.

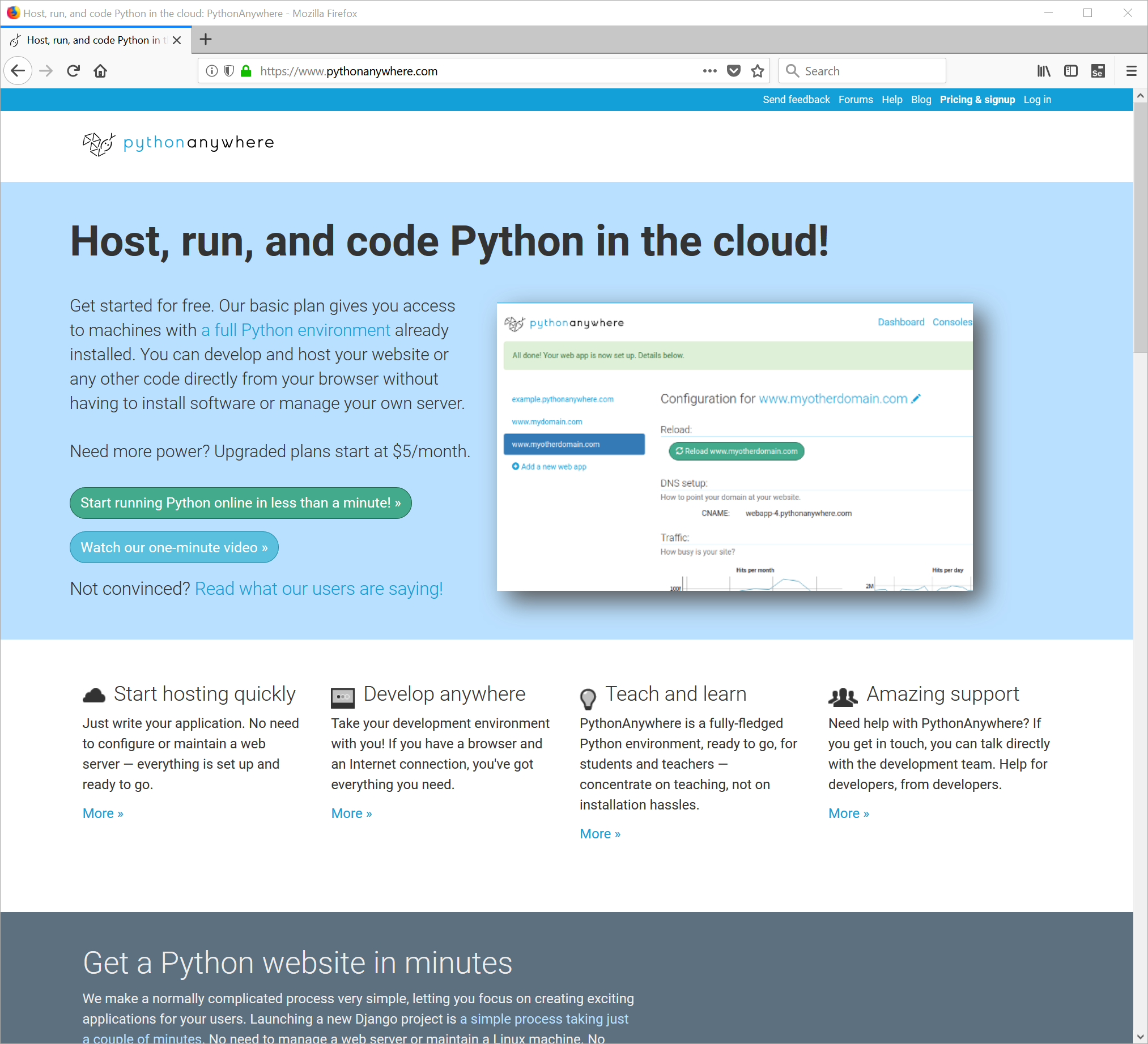
Web link: <https://www.pythonanywhere.com/>

* Via Web Searching

Type or paste “pythonanywhere” into your browser’s search field.

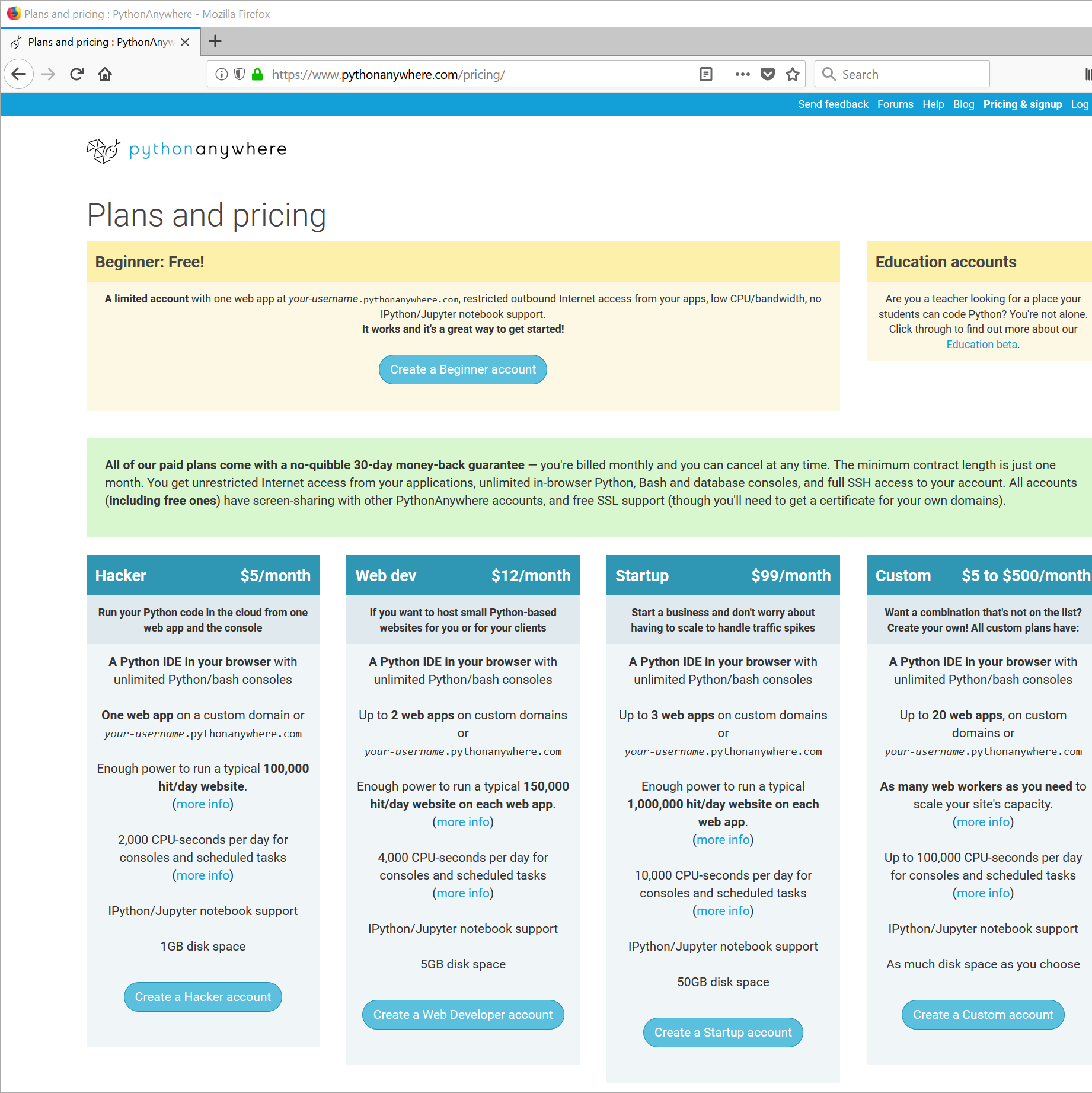
Click on “PythonAnywhere – Official Site”

The PythonAnywhere home site should appear as the figure below.



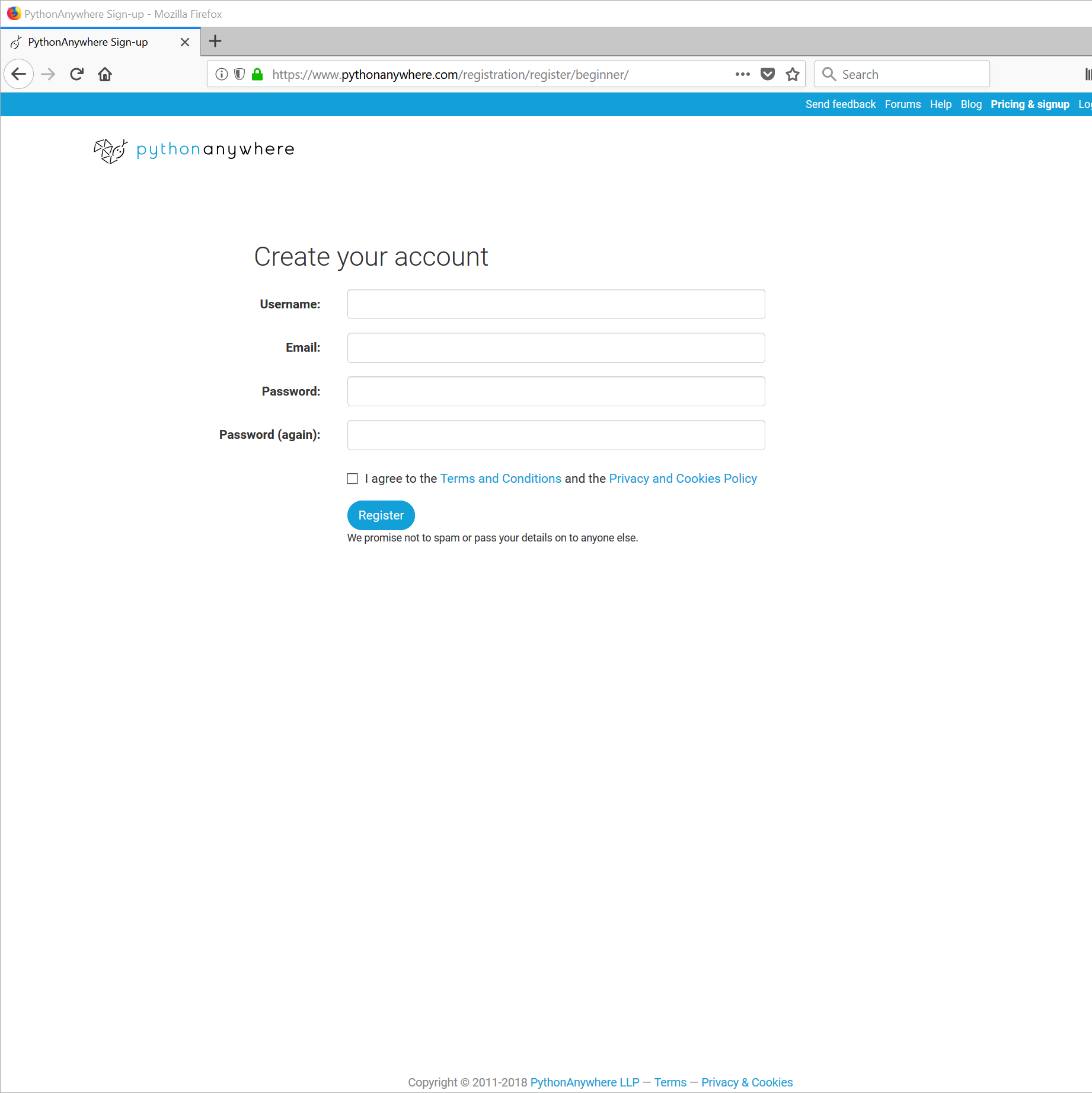
Mouse-Click on the button labeled (Start running Python online in less than a minute)

The following window should appear:



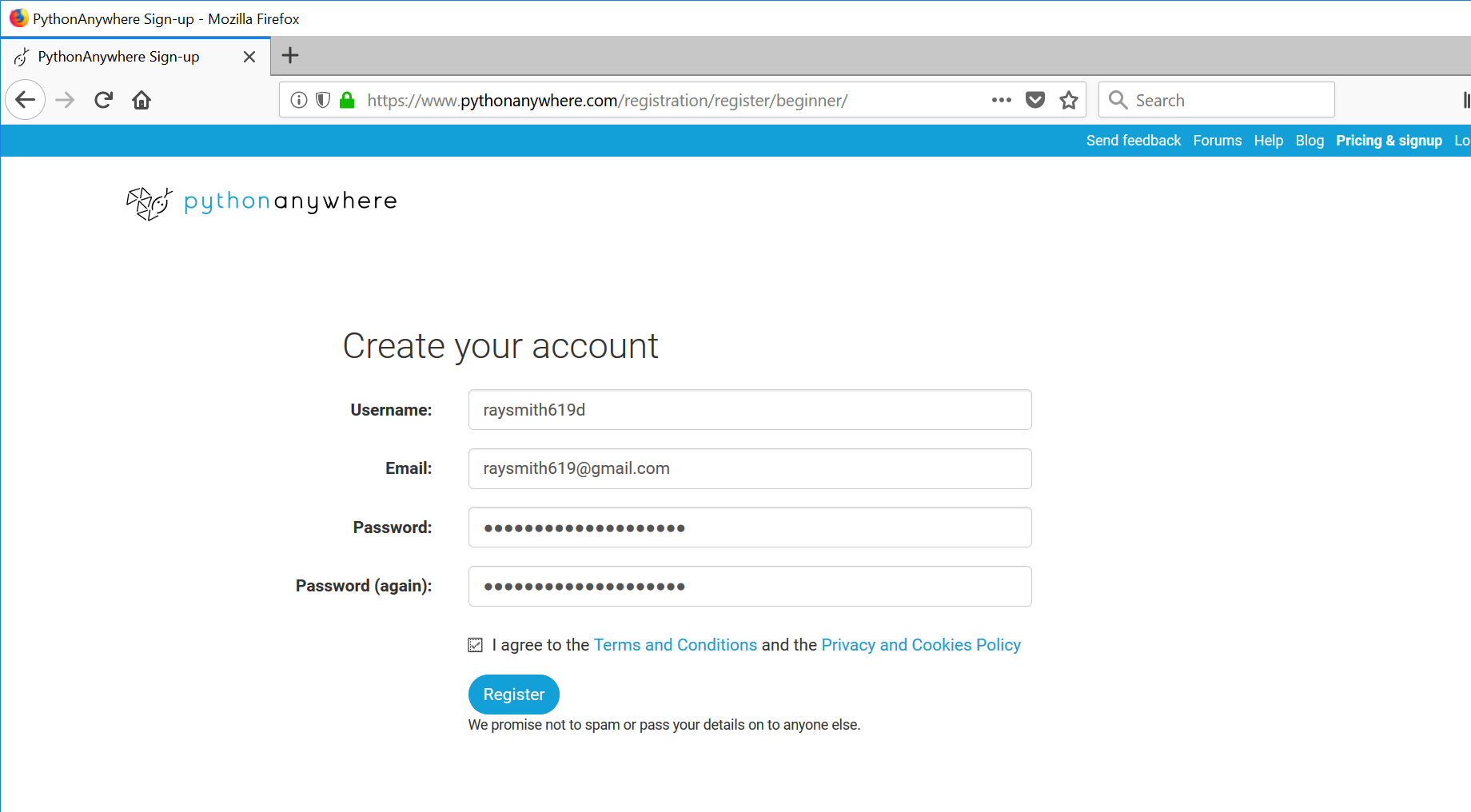
Mouse-Click the button toward the upper middle, labeled (Create a Beginner account)

The following window should appear:



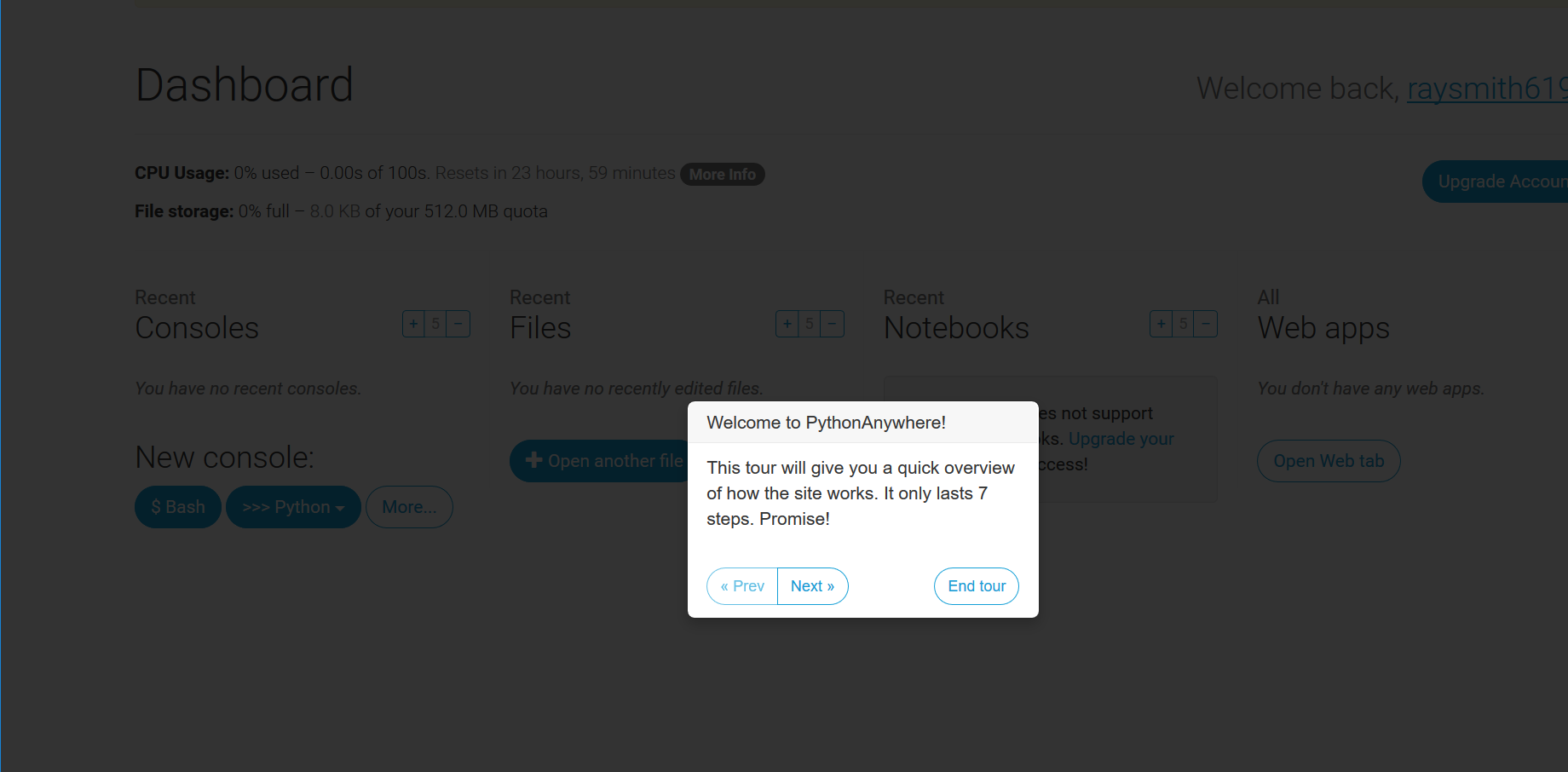
Complete the account entry fields. Please use a password that you can remember and an email address that you will be able to access.

Sample:

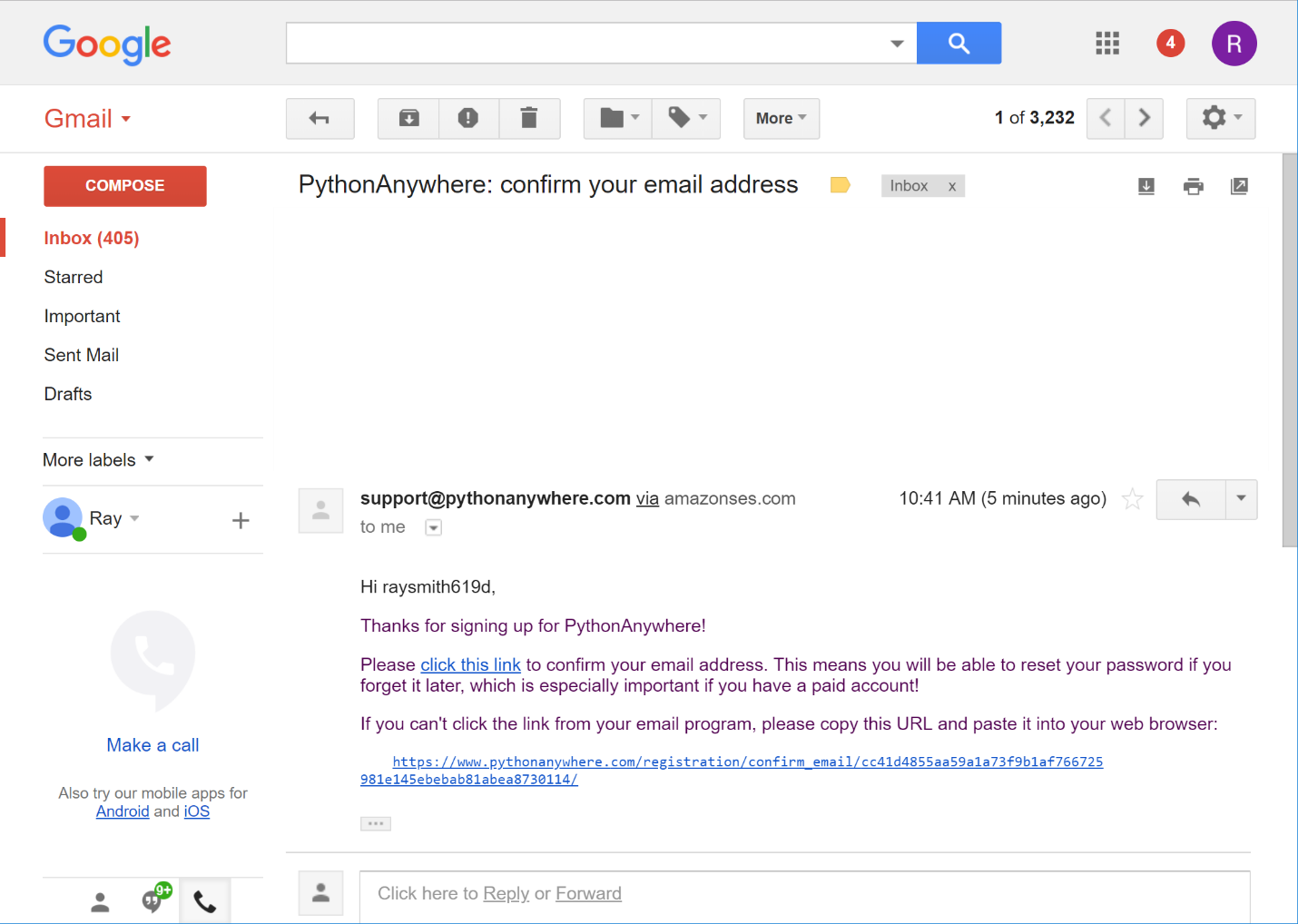


Mouse-Click the button toward the bottom labeled (Register)

The following “Welcome …” dialog box should appear:



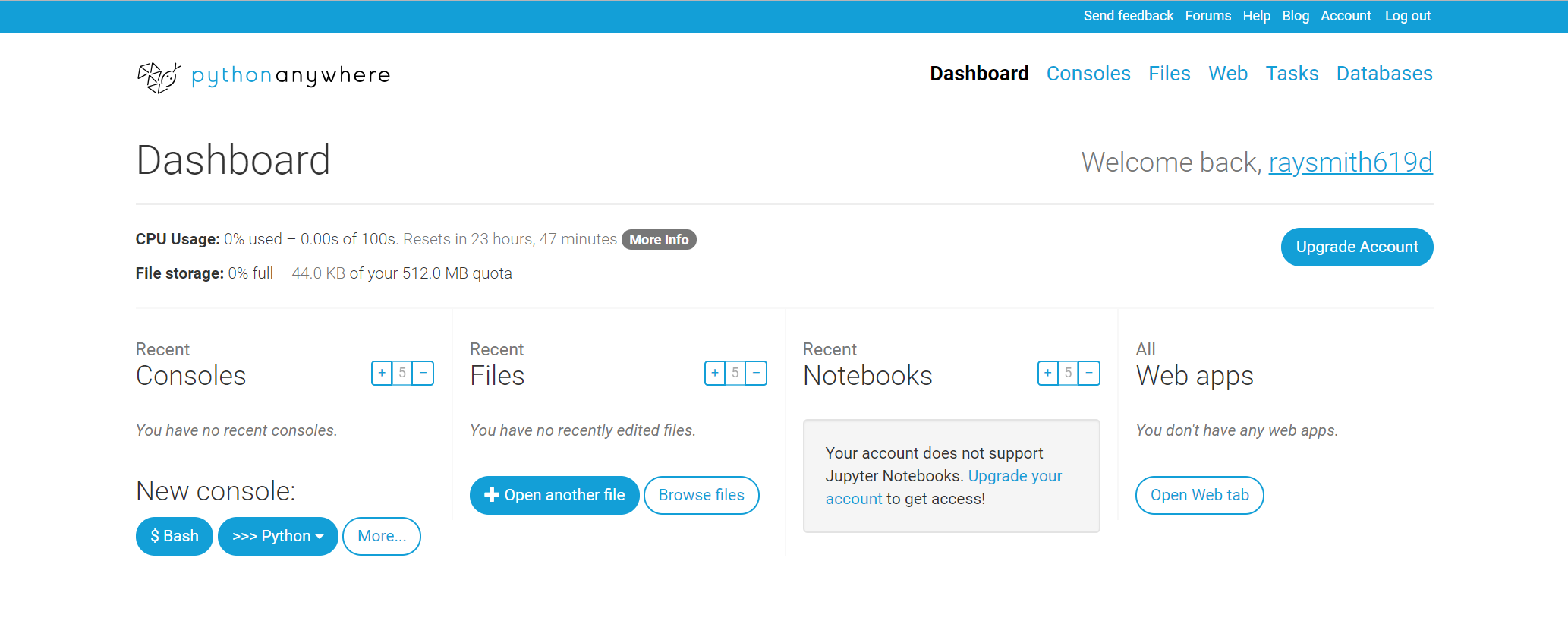
NOTE: You should receive a notification email at your email address supplied similar to the figure below:



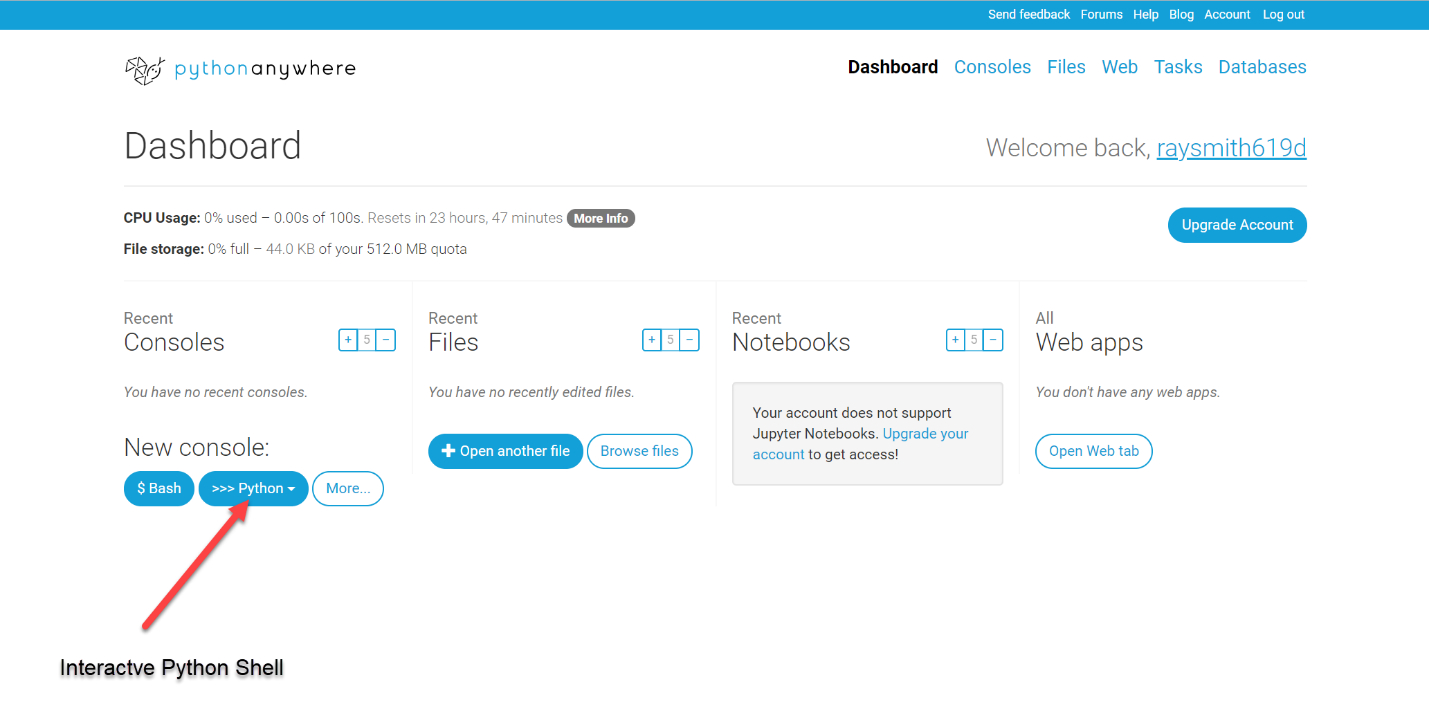
When you receive the email, confirm your email address as requested.

Skip through the tour.

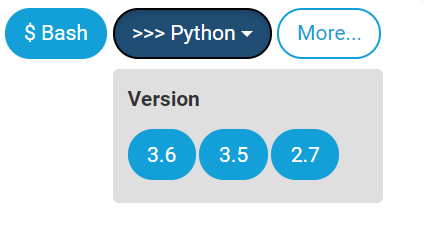
You should end up with the basic PythonAnywhere Dashboard presented in the figure below:



To obtain an interactive Python shell, select the (>>> Python) button as shown in the figure below:

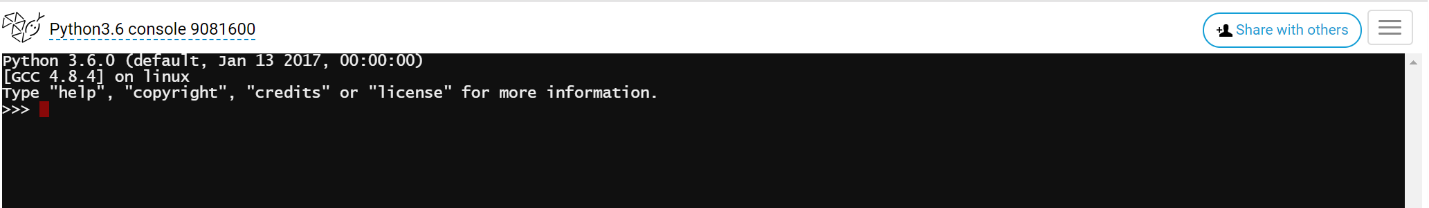


Upon clicking (>>> Python) button, the Version will be requested as shown in the figure below. We will use 3.6.



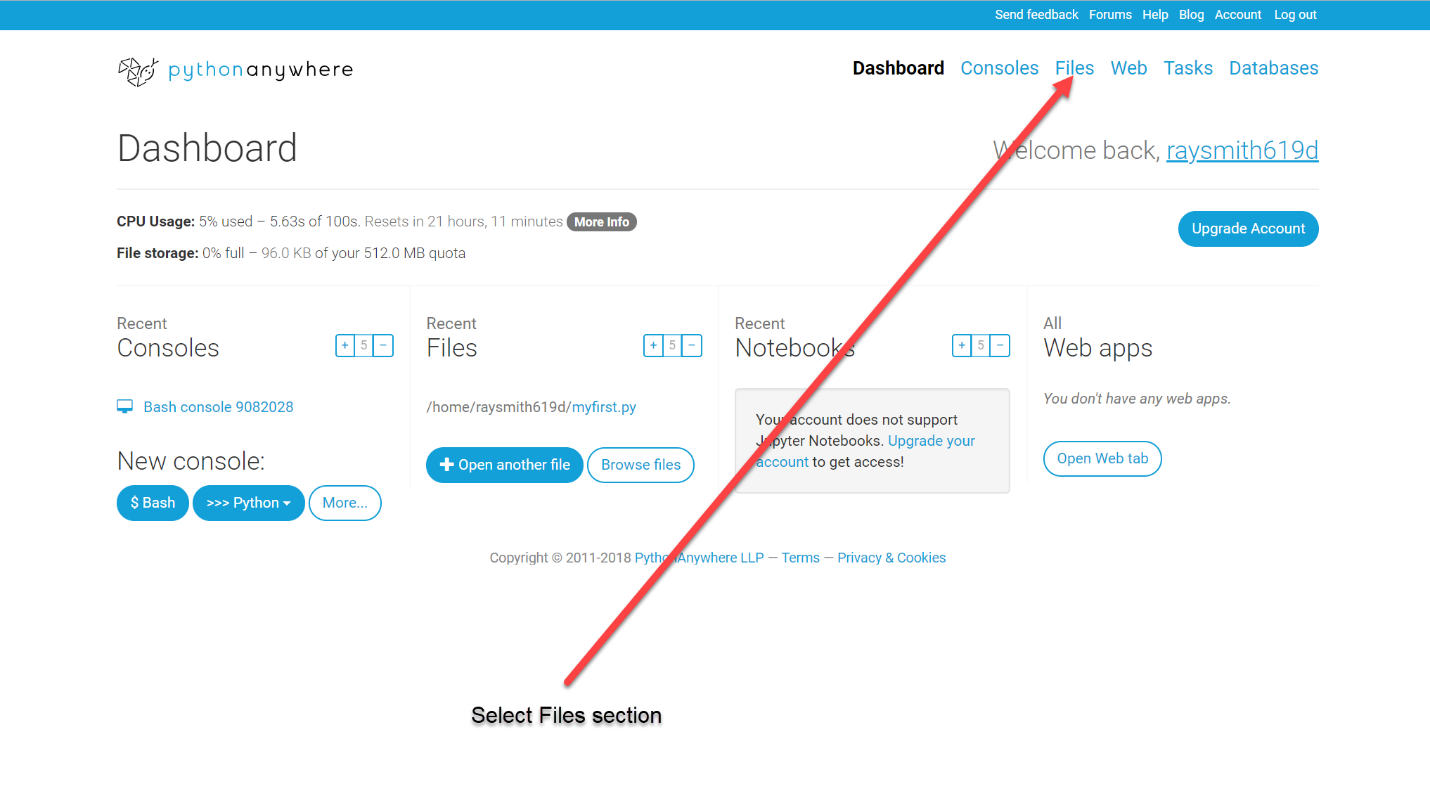
Mouse-click the button labeled (**3.6**)

After a few moments activity a “Python3.6 console” window will appear as in the figure below:

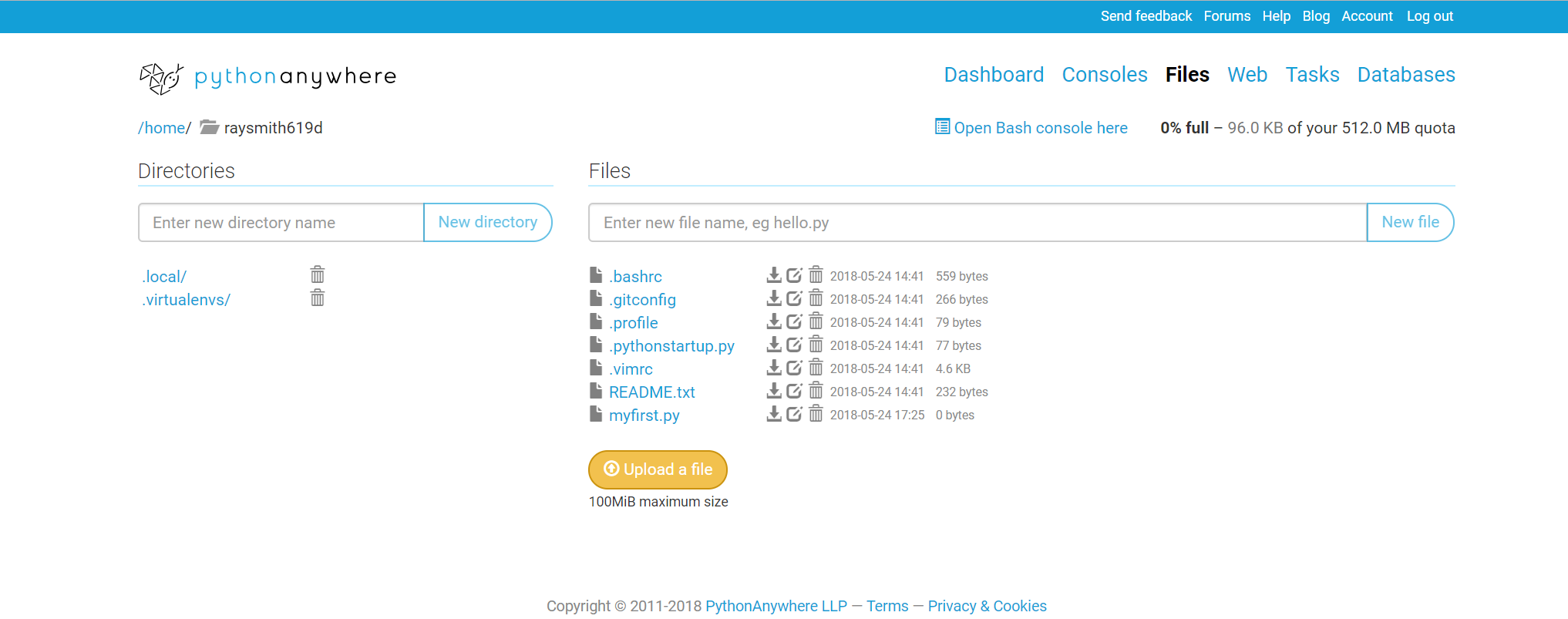


Creating files (for longer programs) in PythonAnywhere:

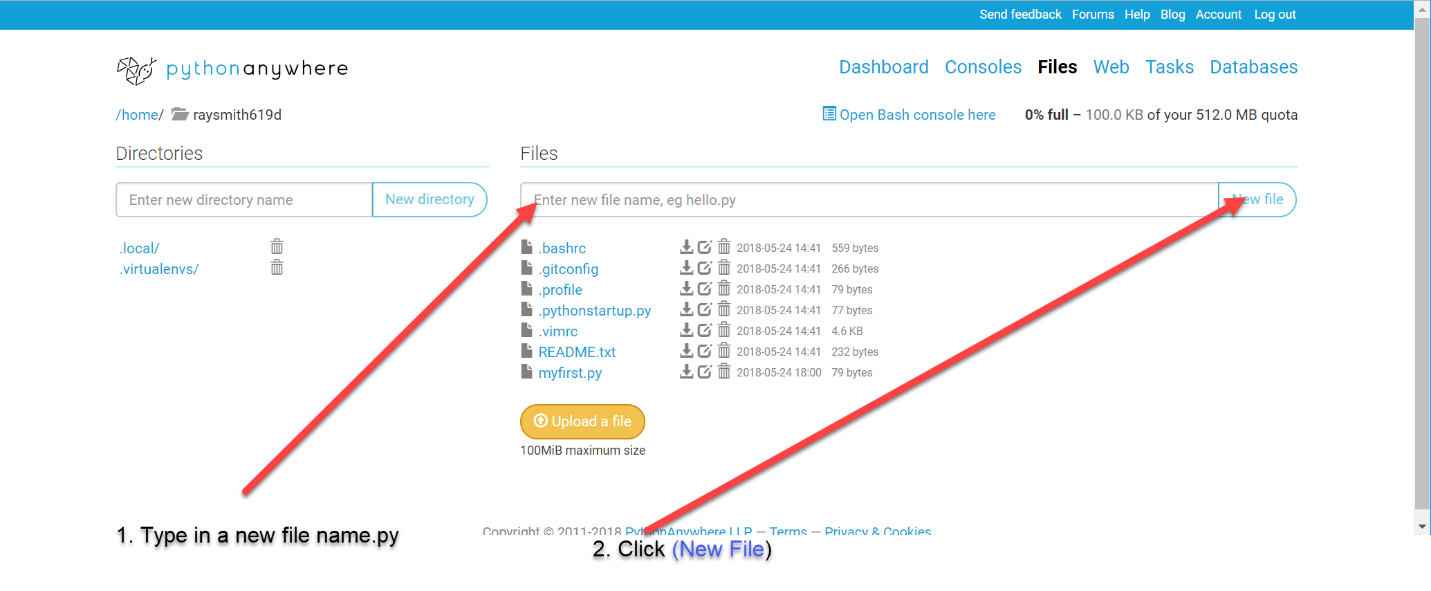
Start in the PythonAnywhere Dashboard window and select the Files section:



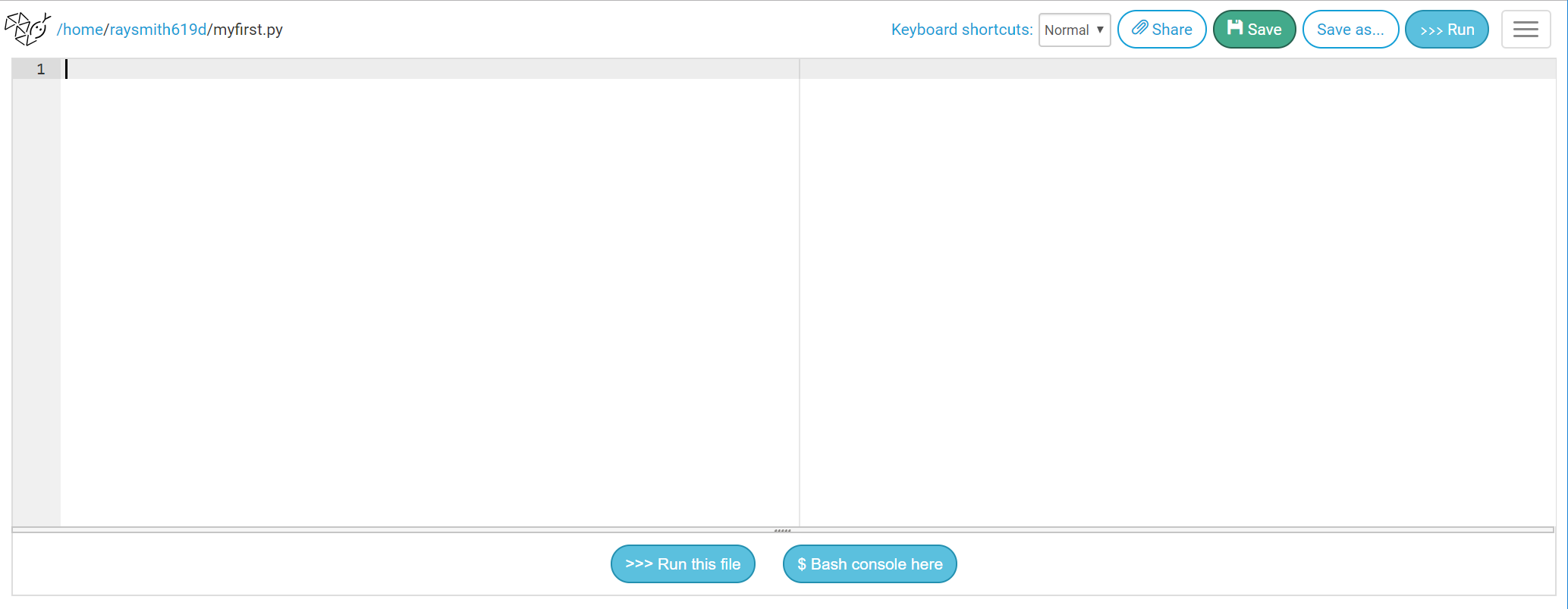
The Files window will appear as below:



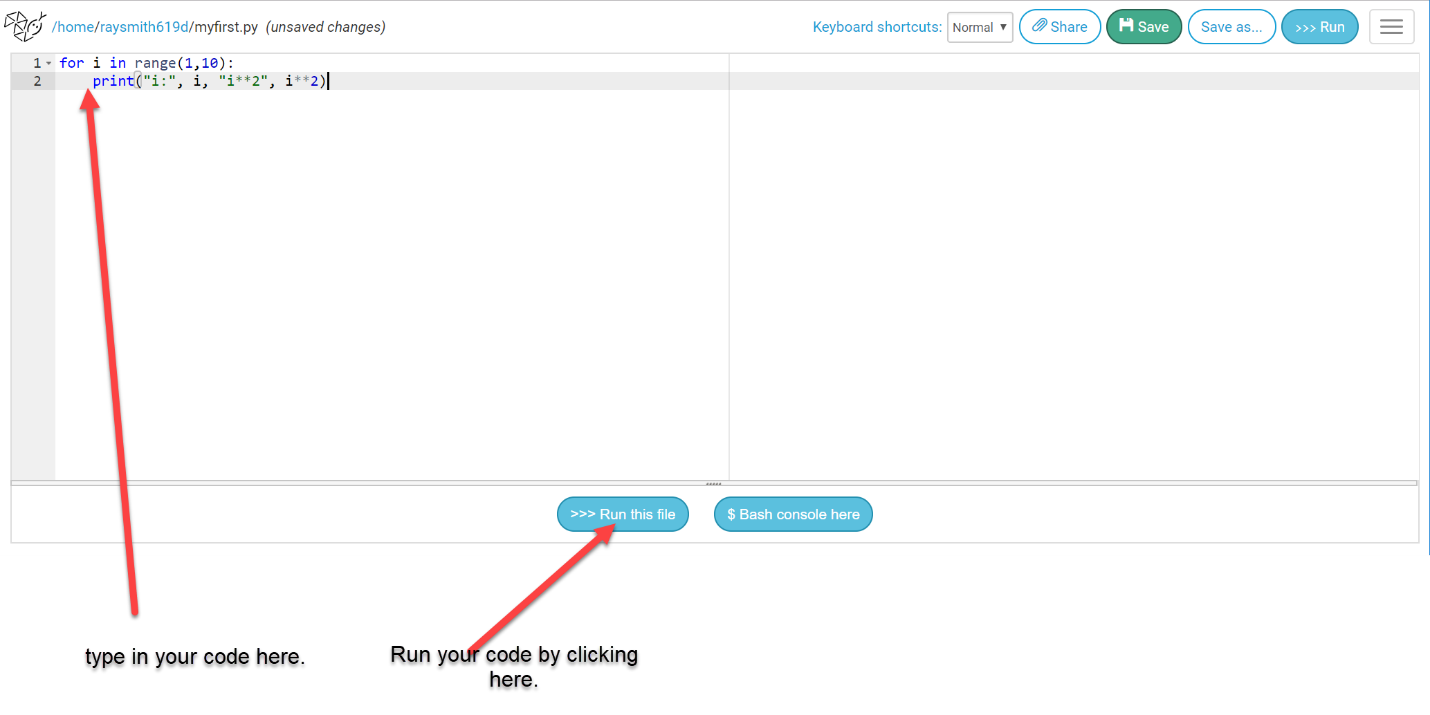
Create a file:



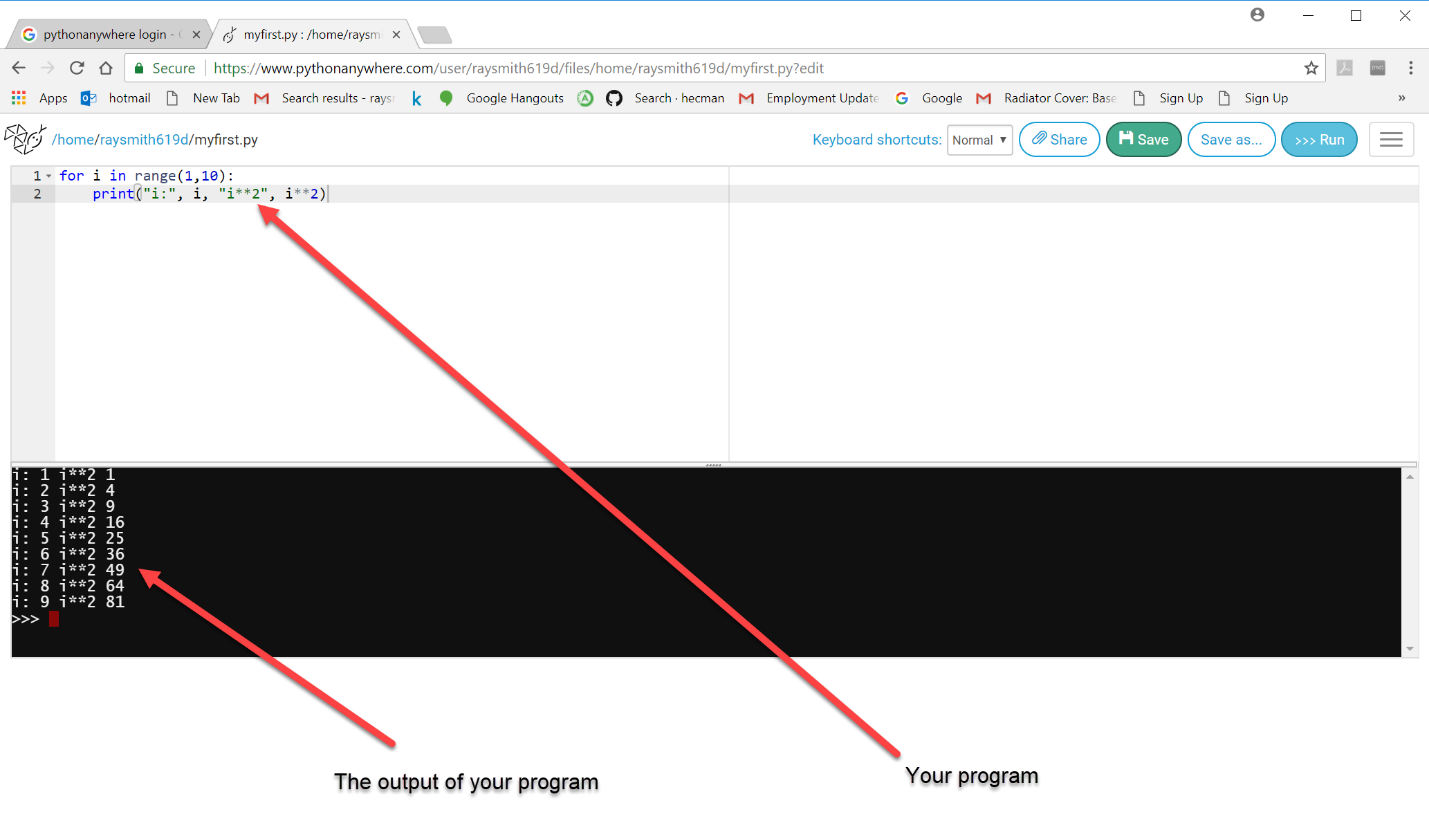
An edit window for this file will appear as below:



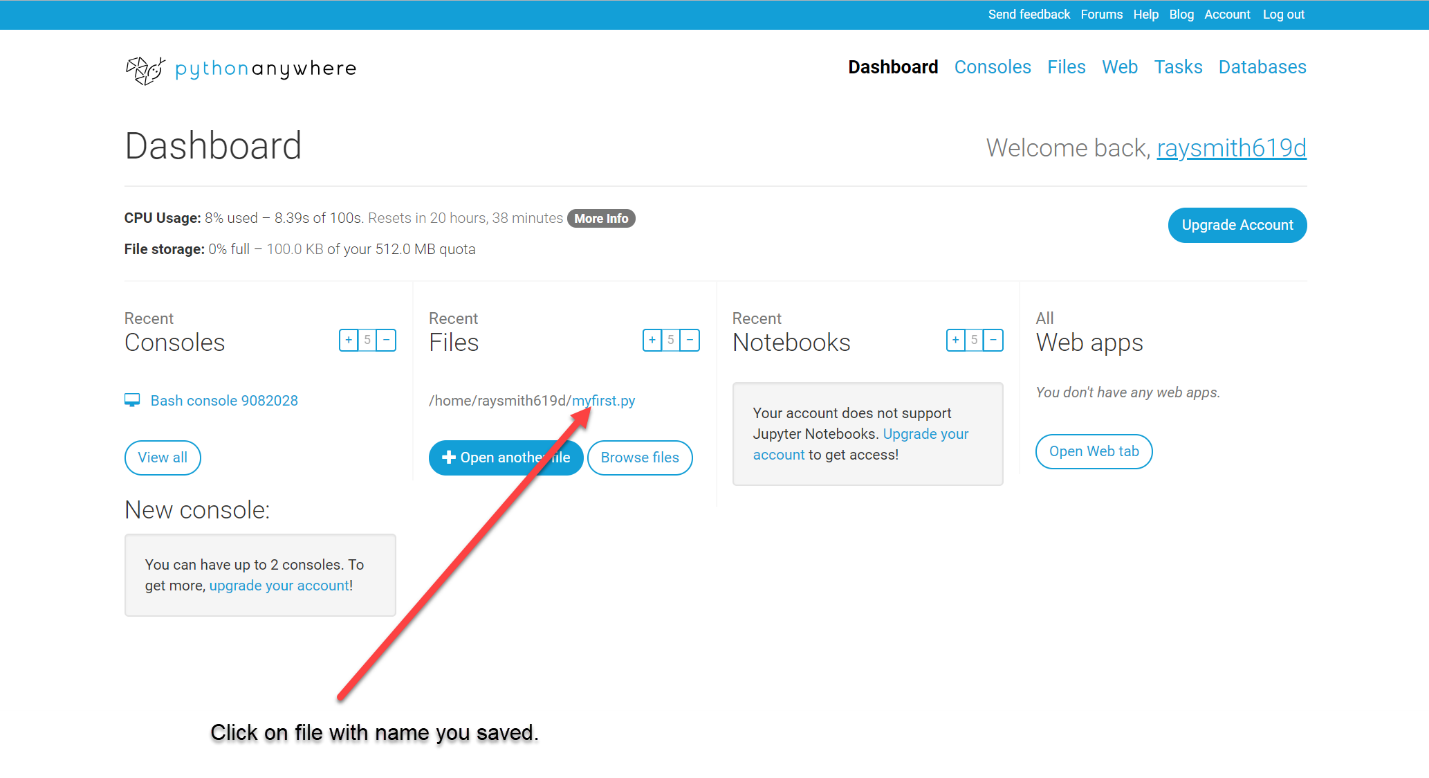
You can enter python code on the left and run this code by clicking the (Run this file) at the bottom.



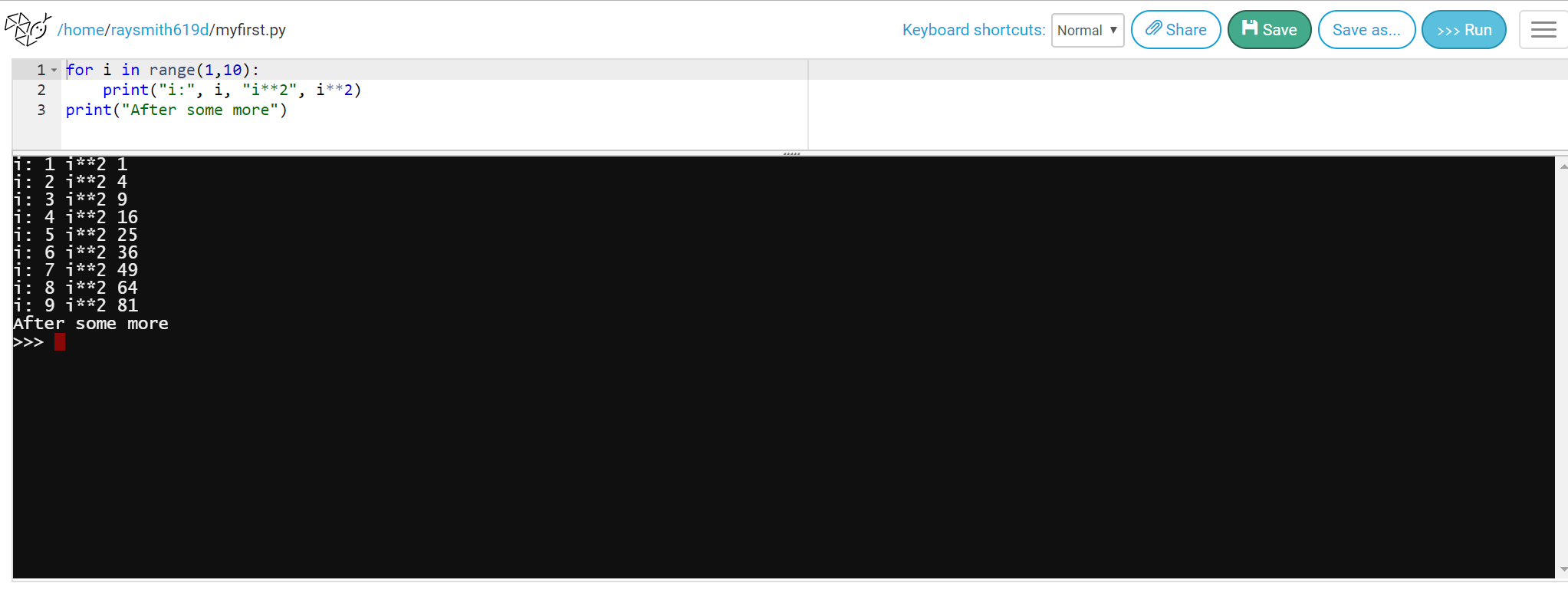
After clicking (Run this file) we see below:

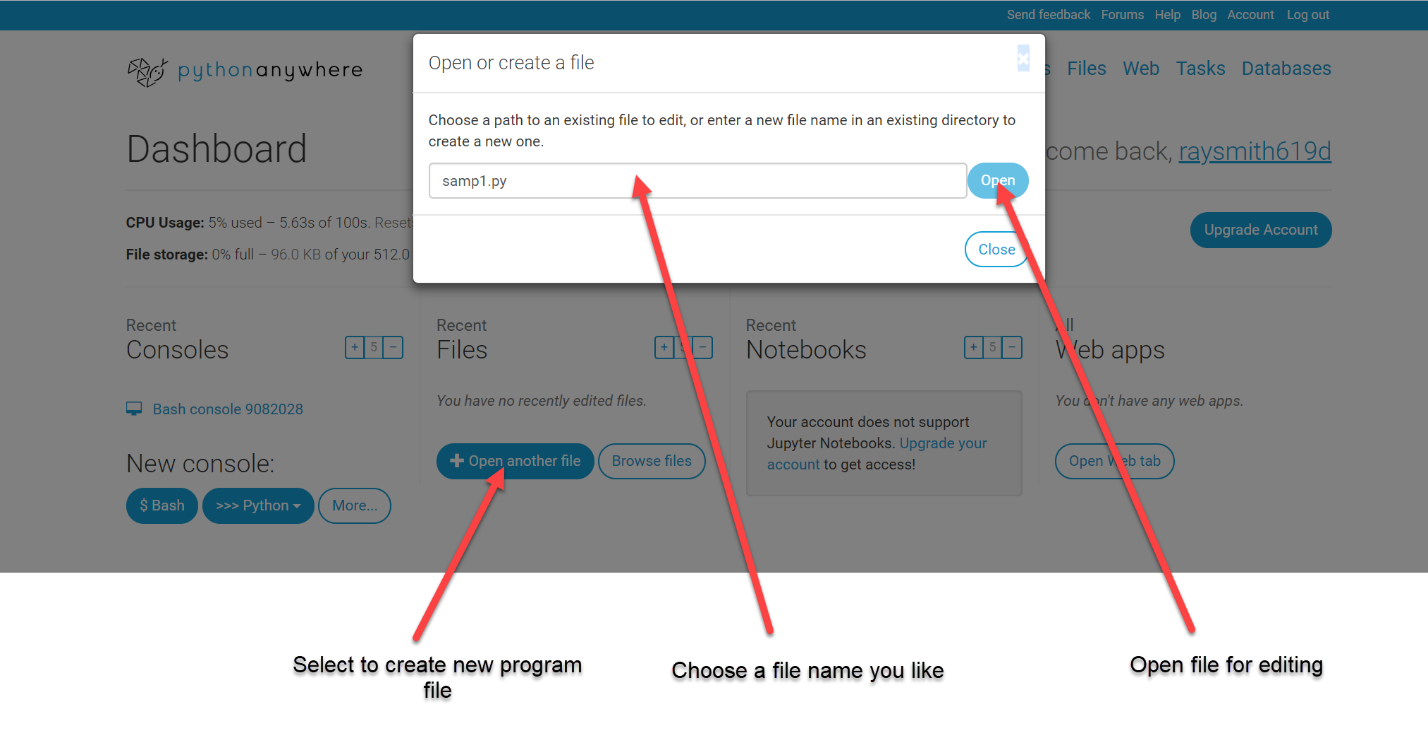


Coming back to a program, after leaving and going to the PythonAnywhere Dashboard:



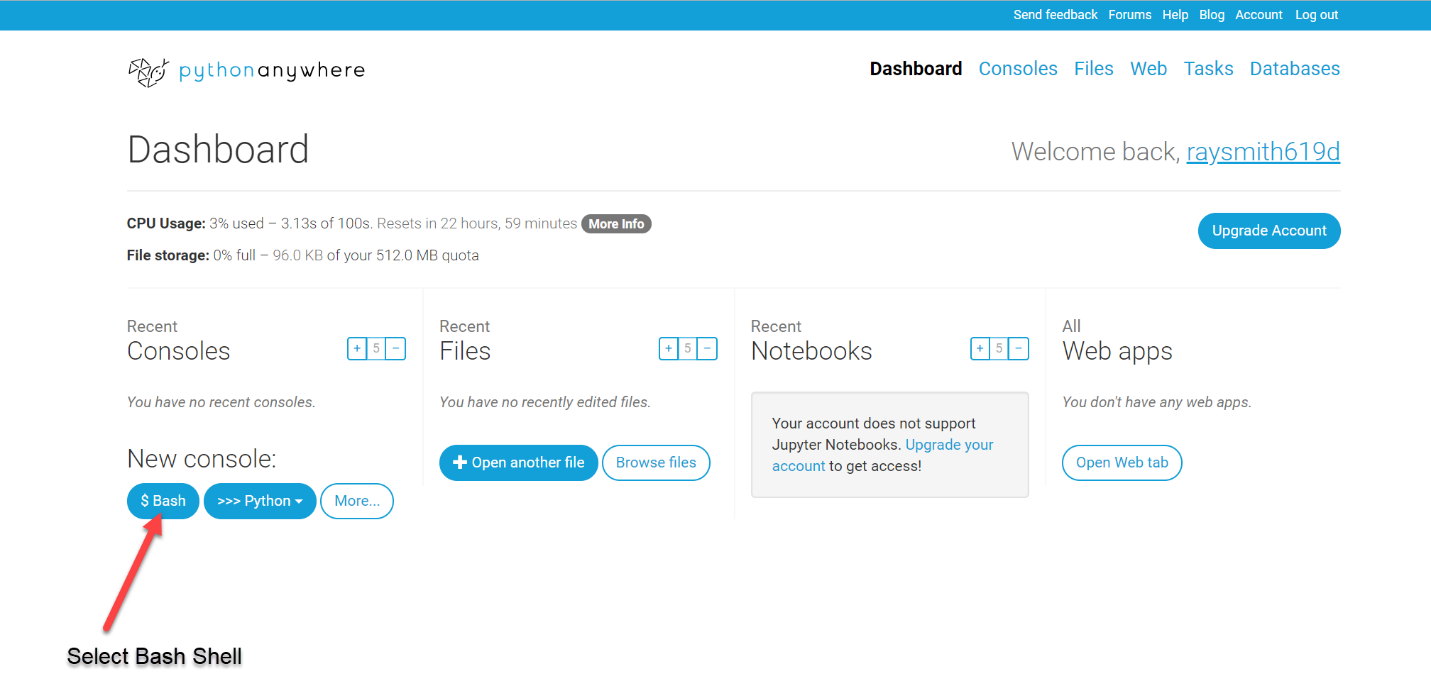
Program file is displayed, plus possibly remembered output, ready for edit and / or running again:



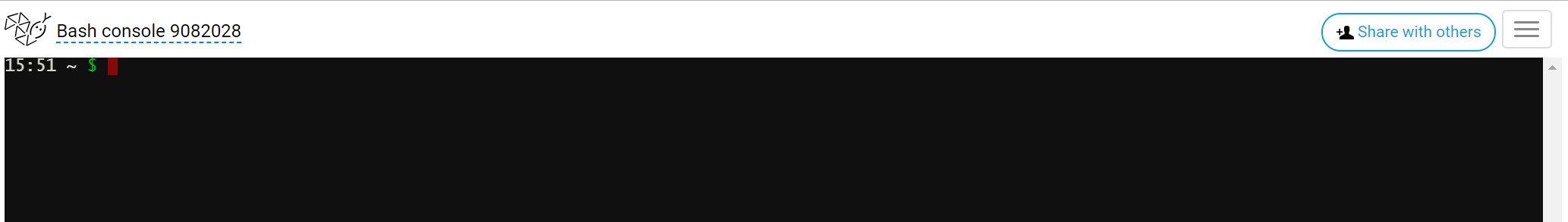


Obtaining Bash Shell (for tools, etc.):

To run a Bash Shell, from the Dashboard window, mouse-click the New console button labeled ($ Bash) as in the figure below:



After a few moments, a window with the Bash console will appear as shown in the figure below:



### Reconnecting to PythonAnywhere Account

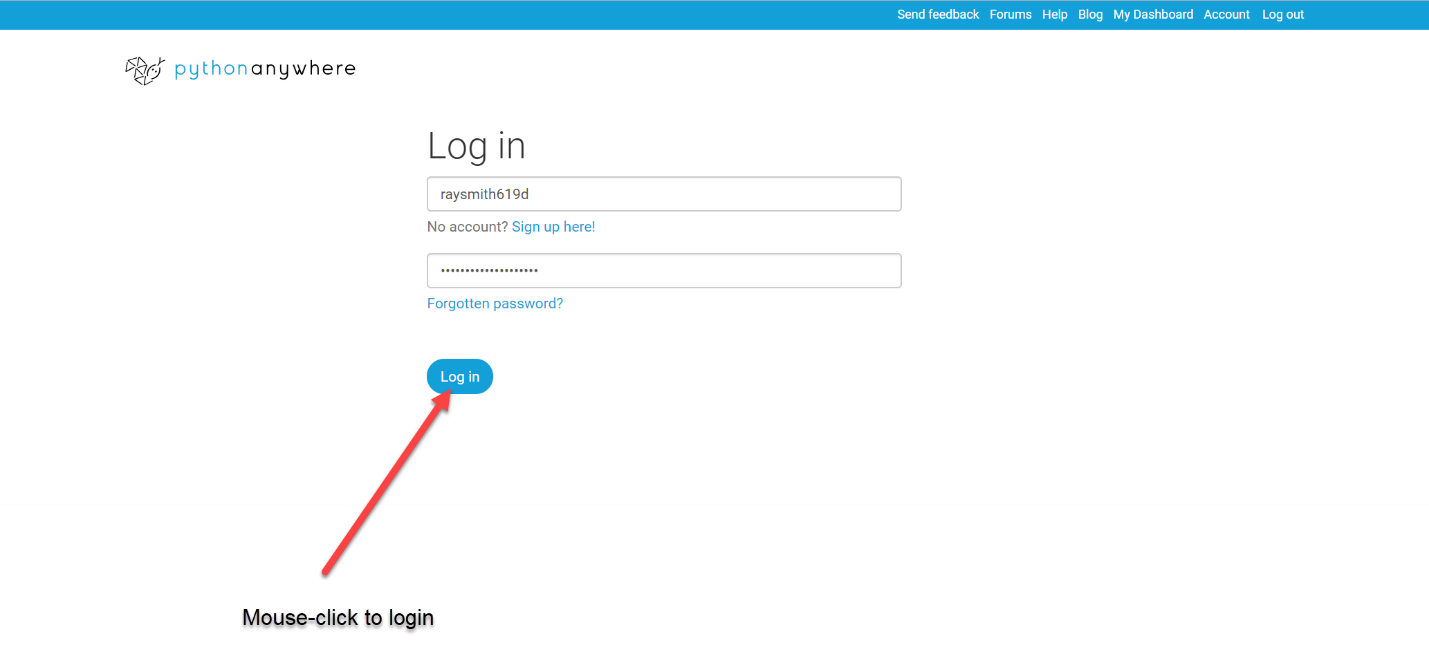
To reconnect to your PythonAnywhere account:

Search / Log in – PythonAnywhere:

[Log in - PythonAnywhere](https://www.pythonanywhere.com/login/?next=/)

<https://www.pythonanywhere.com/login/?next=/>

The following window should appear (Possibly already filled out by your browser):



## Online Python Resources

### PythonAnywhere

<https://www.pythonanywhere.com>

* Need to create an account
* Provides a persistent Linux-like account
* Bash shell

### python.org

<https://www.python.org>

### Other Online Python Resources

#### Jdoodle

<https://www.jdoodle.com/python3-programming-online>

* Need to create an account to save file/project

#### pythonfiddle.com

<http://pythonfiddle.com>

#### onlinegdb

<https://www.onlinegdb.com/>

* Need to create an account to save project
* Has a debugger

#### paiza.io

<https://paiza.io/en/projects/new?language=python3>

<https://paiza.io/en/languages/python3>

#### [amrdraz.github.io/python-debugger/](http://amrdraz.github.io/python-debugger/)

<http://amrdraz.github.io/python-debugger/>

* Simple example- not yet investigated

### Learning Python

#### [Instant Python](http://www.hetland.org/python/instant-python.php) A minimal crash course by Magnus Lie Hetland.

Note that this nice account uses Python2.x so, for the code to work on Python3.x, change the print “……” instances to print(“…..”) instances.