

# Day of Immersion: Programming in the Cloud: Roots of Polynomials

Dr. Jim Newton

February 3, 2025

## 1 Introduction

In this atelier you will experiment with software development in the cloud using GitHub and the Python programming language.

### 1.1 Overview

You will finish the development of a program which will compute the roots of polynomials of degree 1 through 5.

$$\begin{aligned} & ax + b \\ & ax^2 + bx + c \\ & ax^3 + bx^2 + cx + d \\ & ax^4 + bx^3 + cx^2 + dx + e \\ & ax^5 + bx^4 + cx^3 + dx^2 + ex + f \end{aligned}$$

The suite of Python functions you will implement (finish the implementation of) are designed to review your mastery of

- Polynomials
- Factorization
- Quadratic formula
- Finding Extrema using the Derivative
- and more

## 1.2 Flow of Today's Atelier

### 1. Set up your cloud environment:

In Section 2 you will setup your cloud environment. You will then examine and modify a simply `hello world` program.

### 2. Optional: Review of Algebra

Section 4 which explains the math connected to the program you will develop.

### 3. Hello World:

Starting in Section 3, write the *Hello World* program to help you learn how to use the GitHub Code Space programming environment. Complete file `hello.py`.

In the remaining sections, starting with Section 5, you will develop Python code to compute the roots of polynomials.

### 4. Roots of a line:

In Section 5 you'll find the sole root of a 1st degree polynomial (called a line). Complete file `line.py`. Use Section 5 to understand the theory.

### 5. Roots of a quadratic:

You'll complete the file `quadratic.py`. You'll find at most two roots of a 2nd degree polynomial (called a quadratic). Use Section 6 to understand the theory.

### 6. Roots of a cubic:

In Section 7, you'll find at most three roots of a 3rd degree polynomial (called a cubic). Complete the code in `cubic.py`.

### 7. Roots of quartic and quintic:

If there's time remaining in the atelier, Sections 8 and 9 deal with 4th and 5th order polynomials. Even if there's not enough time to finish all the exercises in this atelier, you may continue the work on your own time, because your account of GitHub will remain in place as long as you do not delete it.

Good Luck! And Happy Coding.

## 2 Programming in the Cloud with GitHub

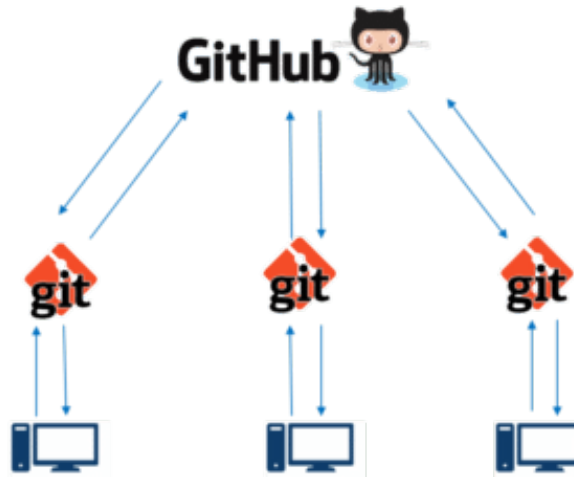


Figure 1: GitHub is a cloud service that hosts code repositories which you can access via a web browser.

### 2.1 Objectives

The assignment in this section is:

1. Create a GitHub account
2. Open the `src/hello.py` with GitHub Code-Spaces.
3. Run the *tests* in `tests/test_hello.py` using Code-Spaces

### 2.2 Overview

For this atelier you will use **GitHub CodeSpaces** as your development environment. This is a cloud-based interactive development environment (IDE). If you are interested in continuing to work on the project after the atelier is finished today, you will still be able to access GitHub CodeSpaces for as long as you wish in the coming days, months, years.

You will start with existing code in a central repository of code, hosted by GitHub on the cloud. You will download (clone) a copy of this repository, edit existing code, and create new code.

GitHub is a web based service which hosts thousands of projects, small and large, in the cloud. Typically, users have a development environment on their

local computer (laptop or desktop) and periodically upload and download changes to the cloud.

The URL <https://github.com/jimka2001/immersion>, is the public entry point to the GitHub project which we will work on in this atelier.

You must create an account on GitHub (if you don't have one already). This will be done in Section 2.3.

## 2.3 Getting Started

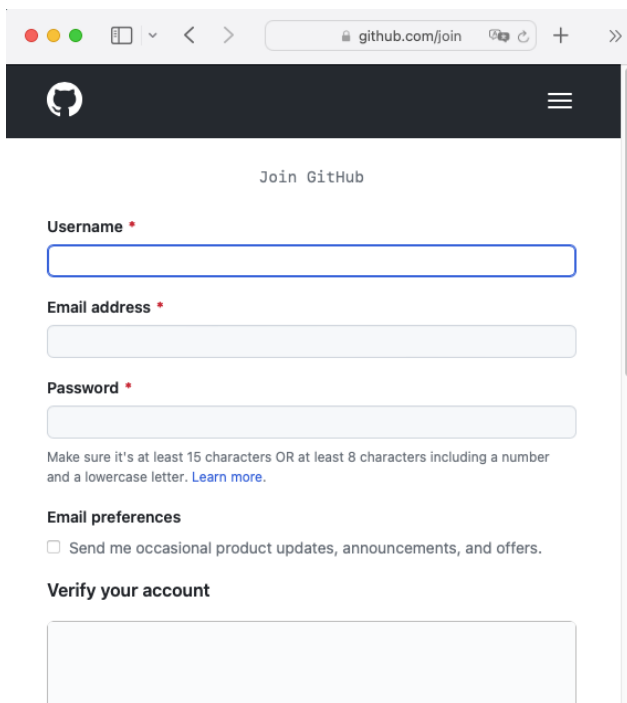
We recommend you use the Chrome web browser.



We need to set up the development environment. The following steps should help.

## 2.4 Account Creation

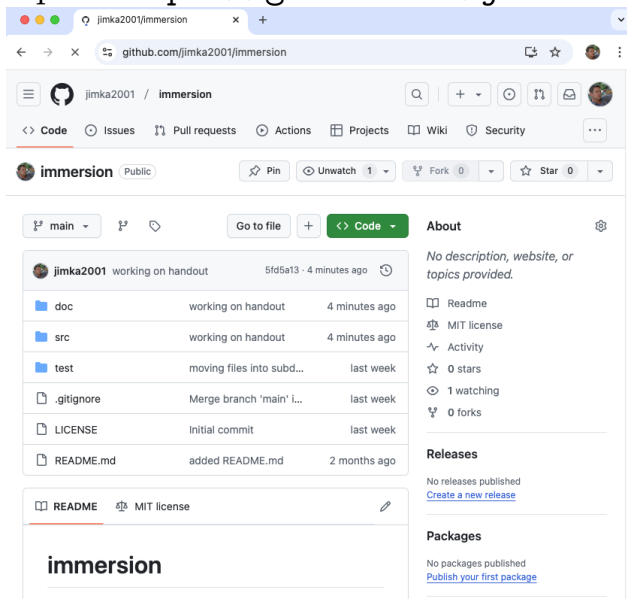
Create a GitHub account using an abstract user name. Don't use your real name. Instead use an artificial name use as your name from Instagram or other social media. You may also make up a completely new name as long as no other person has already used that name on GitHub. <https://github.com/join>. To complete the process of creating a GitHub account you may need to authenticate using your mobile phone.



The screenshot shows the GitHub registration page. At the top is the GitHub logo and a hamburger menu. Below is the heading "Join GitHub". The form includes three required fields: "Username", "Email address", and "Password", each with a red asterisk. Below the password field is a note: "Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)". There is an "Email preferences" section with a checkbox for "Send me occasional product updates, announcements, and offers." and a "Verify your account" section with a large empty box for a verification code.

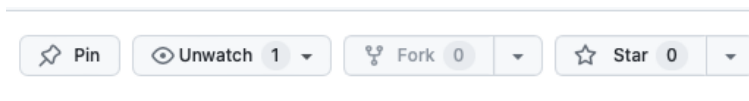
## 2.5 Open the Repository

Open <https://github.com/jimka2001/immersion> with your web browser.



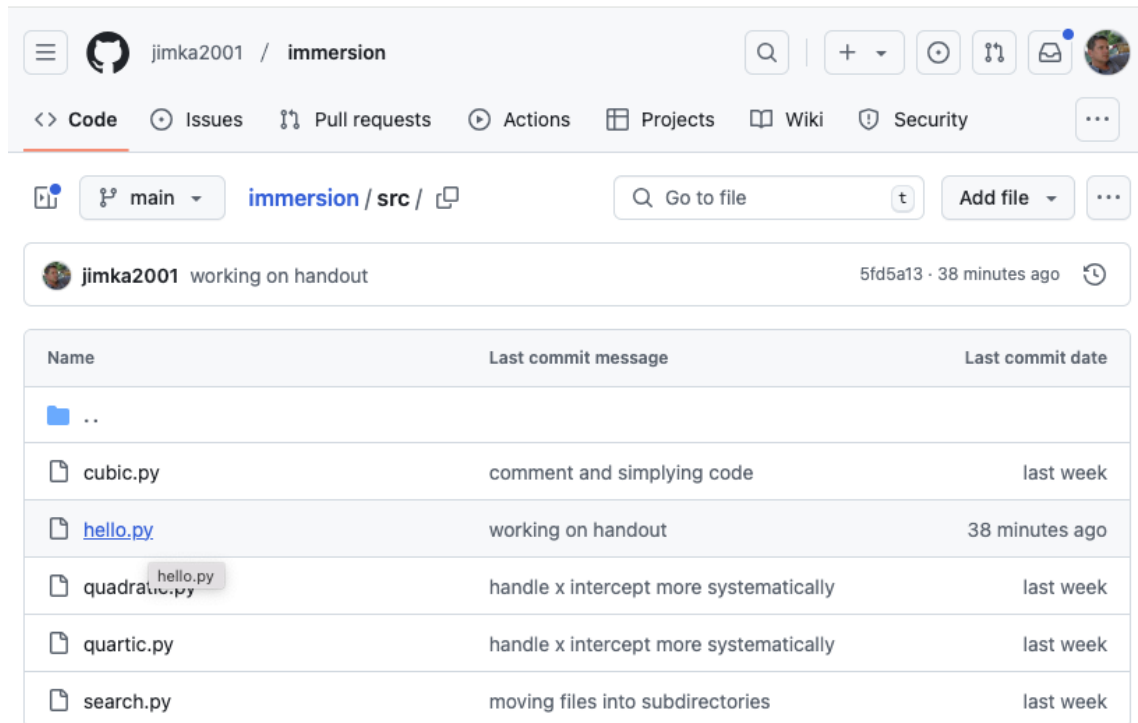
## 2.6 Fork yourself a copy

Fork the repository. This gives you a private copy. You will make changes necessary changes in the code using this private copy.

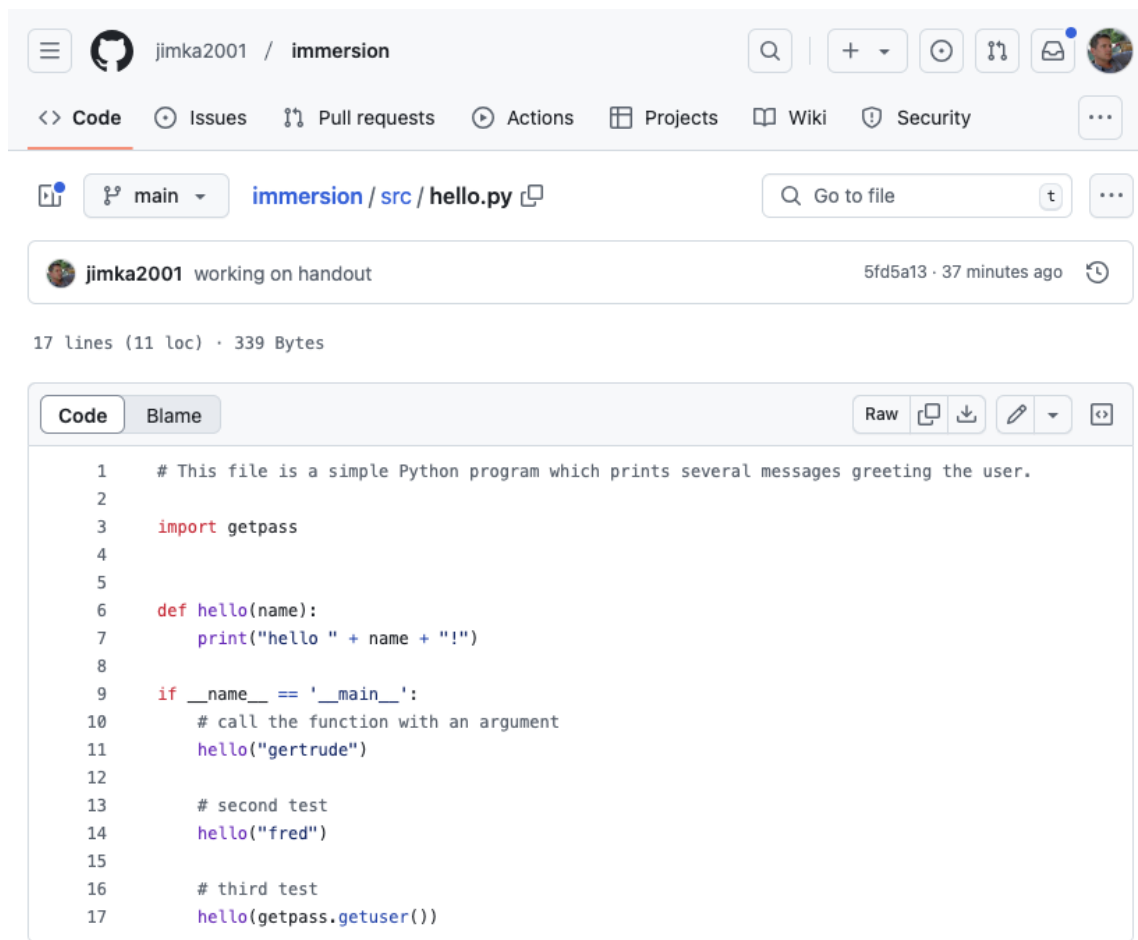


## 2.7 Open a Python File in Code-Spaces

1. Navigate to `src/hello.py` to see something like this.



2. Click `hello.py` to open the file in an editor pane. You should see something like what is shown here:



immersion / src / hello.py

Go to file

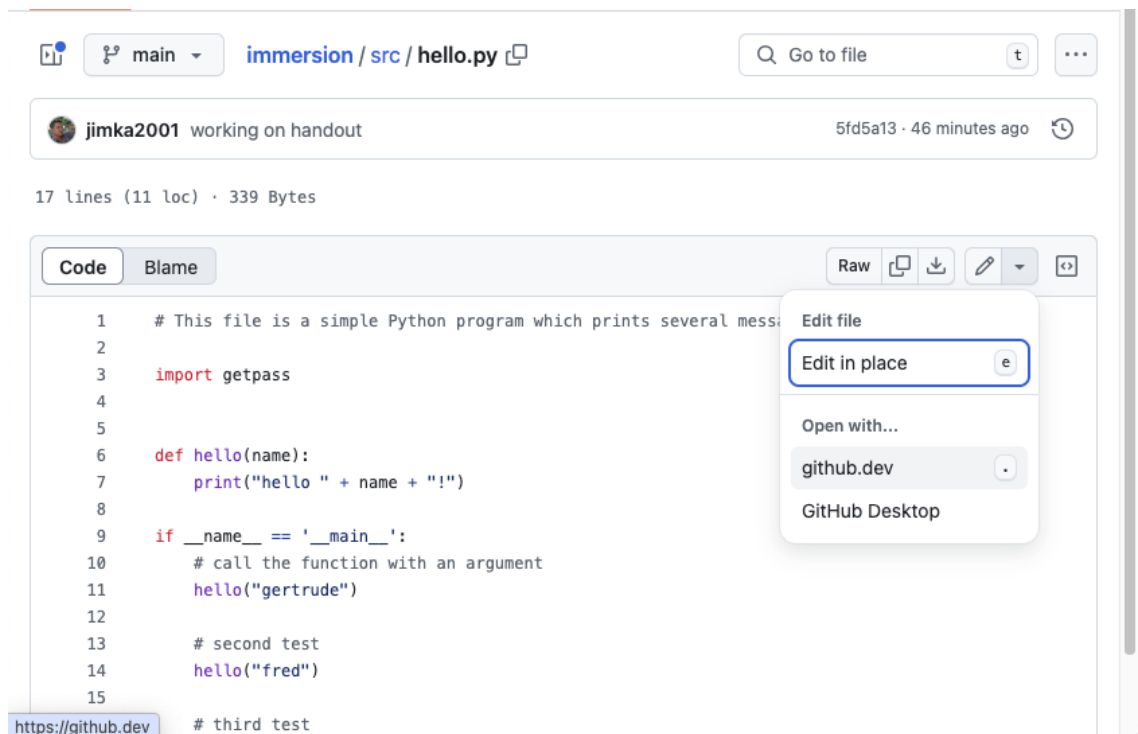
jimka2001 working on handout 5fd5a13 · 37 minutes ago

17 lines (11 loc) · 339 Bytes

Code Blame

```
1 # This file is a simple Python program which prints several messages greeting the user.
2
3 import getpass
4
5
6 def hello(name):
7     print("hello " + name + "!")
8
9 if __name__ == '__main__':
10     # call the function with an argument
11     hello("gertrude")
12
13     # second test
14     hello("fred")
15
16     # third test
17     hello(getpass.getuser())
```

3. Open the file by clicking `github.dev` not `Edit in place`.



immersion / src / hello.py

Go to file

jimka2001 working on handout 5fd5a13 · 46 minutes ago

17 lines (11 loc) · 339 Bytes

Code Blame

```
1 # This file is a simple Python program which prints several mess
2
3 import getpass
4
5
6 def hello(name):
7     print("hello " + name + "!")
8
9 if __name__ == '__main__':
10     # call the function with an argument
11     hello("gertrude")
12
13     # second test
14     hello("fred")
15
16     # third test
```

Edit file

Edit in place e

Open with...

github.dev

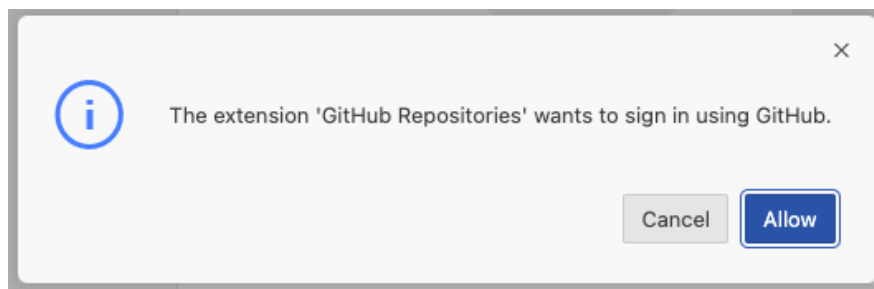
GitHub Desktop

<https://github.dev>

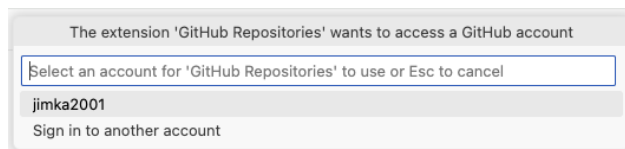
4. You may need to wait while setting up. If this setup seems to never finish, it may mean you need to use Chrome as your web browser.



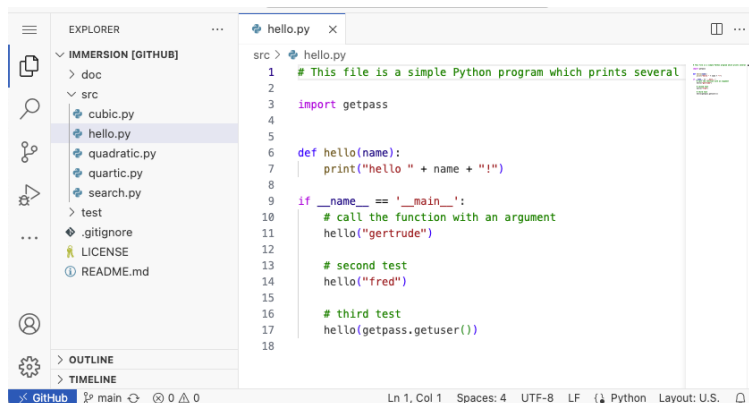
5. GitHub may ask for permission to see your repository. Click on **Allow**.



6. GitHub may ask for your GitHub ID. Normally you just select the default one.



7. Finally, the GitHub development environment should be opened.



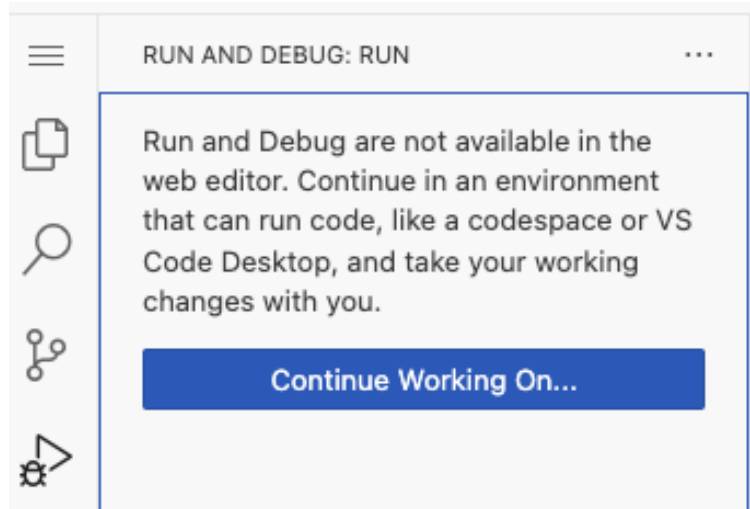
## 2.8 Set up the Editor

1. Now you can edit your code but you cannot run nor test it.

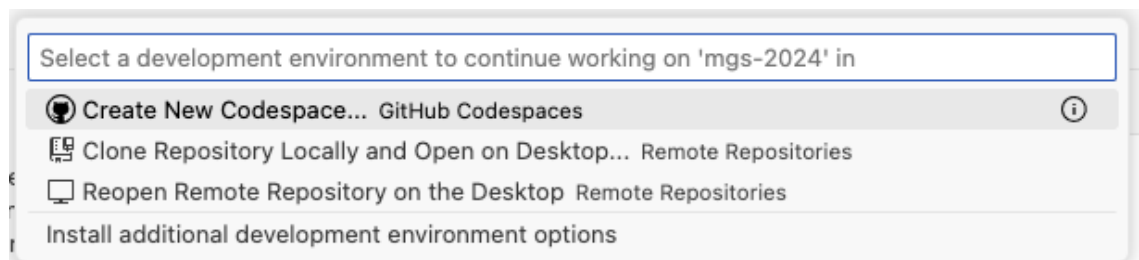




2. Find the icon on the left hand side of the browser window. Press that icon to see the following message.

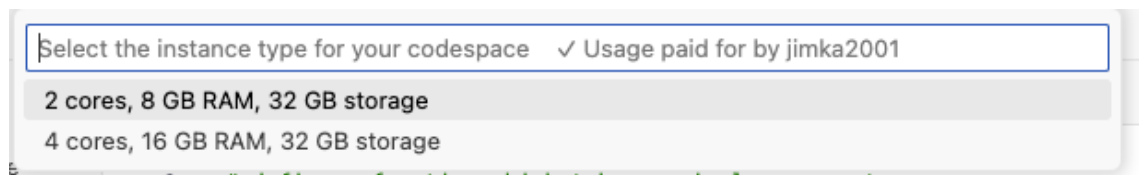


3. Press continue, and you should see a prompt such as the following to create a code space.



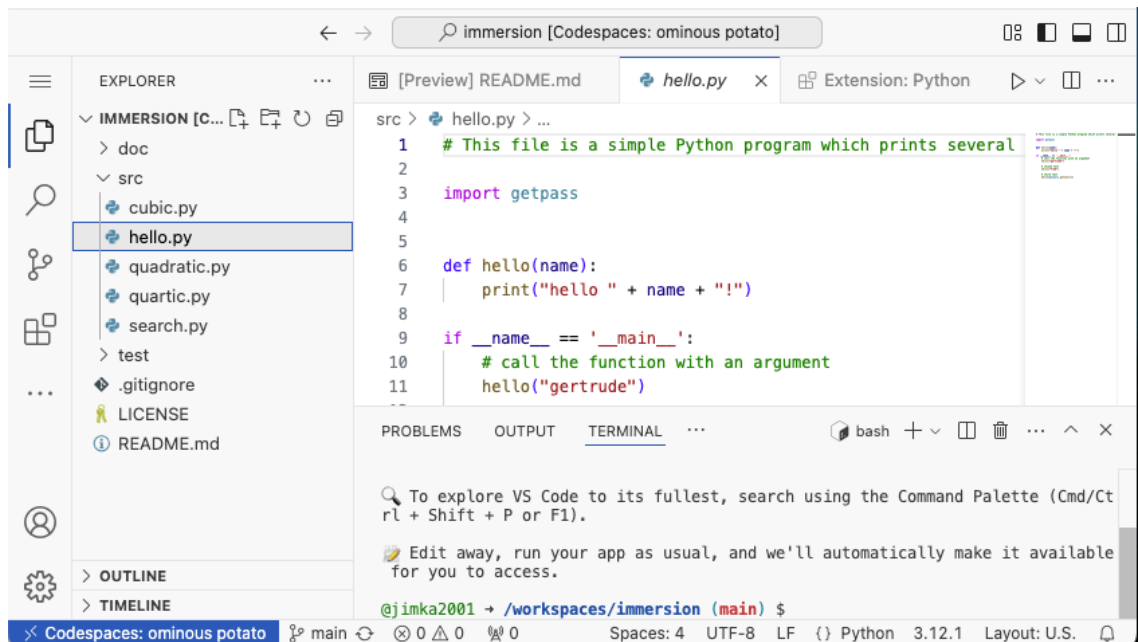
4. Select **Create New Codespace... GitHub Codespaces**.

5. You may be asked how many cores do you want.

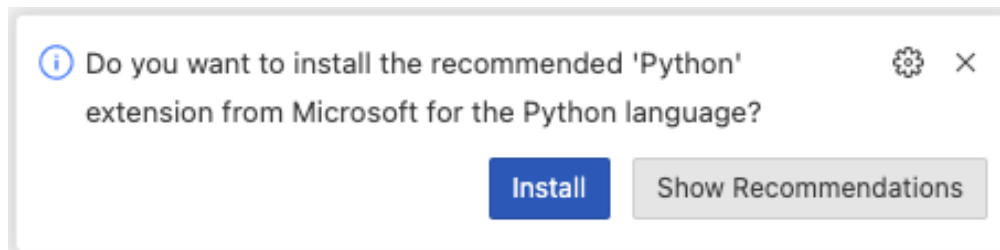


6. You should select the minimum: **2 cores, 8 GB RAM, 32 GB storage**.

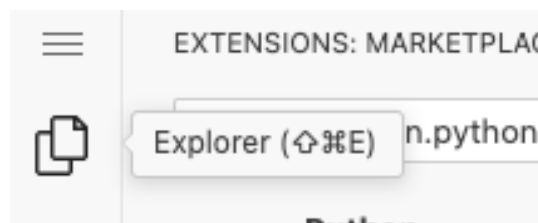
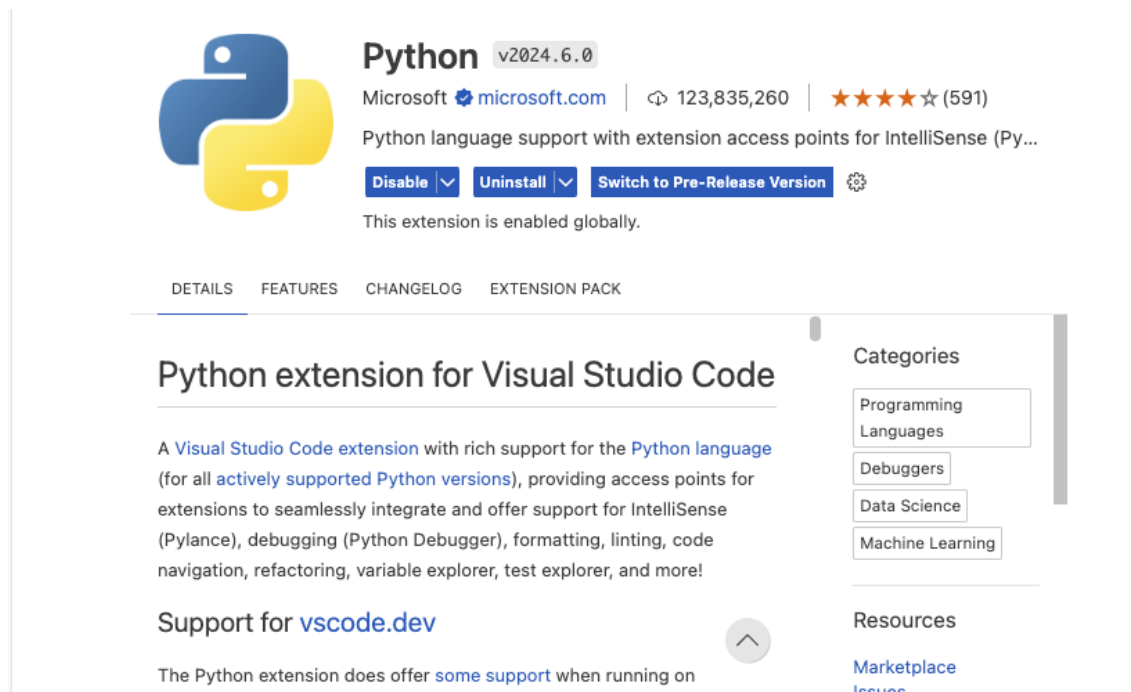
7. You'll probably now need to reopen the **src/hello.py** file.



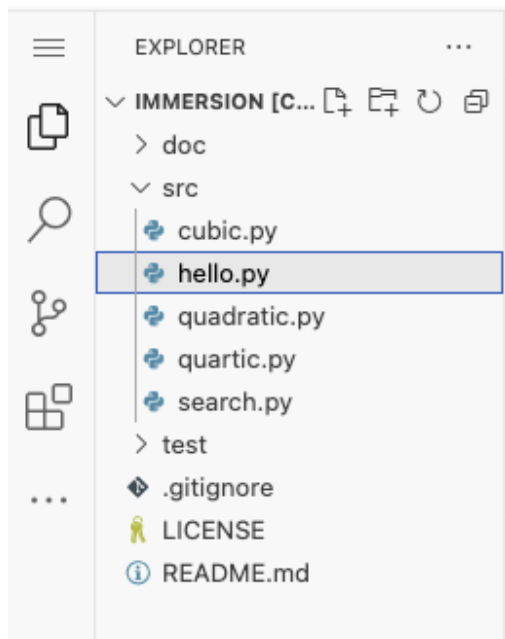
8. You may be asked to install the Python extension. Press **Install**.



9. Wait until it finishes installing. When it finishes installing, you should see something like this:



10. Reopen the explorer: `src/hello.py` file.



## 3 Hello World

The first program we usually write in any programming language is called `hello-world`. It is a very simple program. However getting it to run means you have to understand how to use the programming environment.

### 3.1 Examine the Code

Take a look at the Python code in the file `src/hello.py`. In particular there are two sections in the code,

1. A declaration of the function named `hello`, shown in Listing 3.1
2. Two conditional calls to the function `hello`, each time with a different argument, shown in Listing 3.2.


**Listing 3.1** (*Declaration of Function `hello`*)

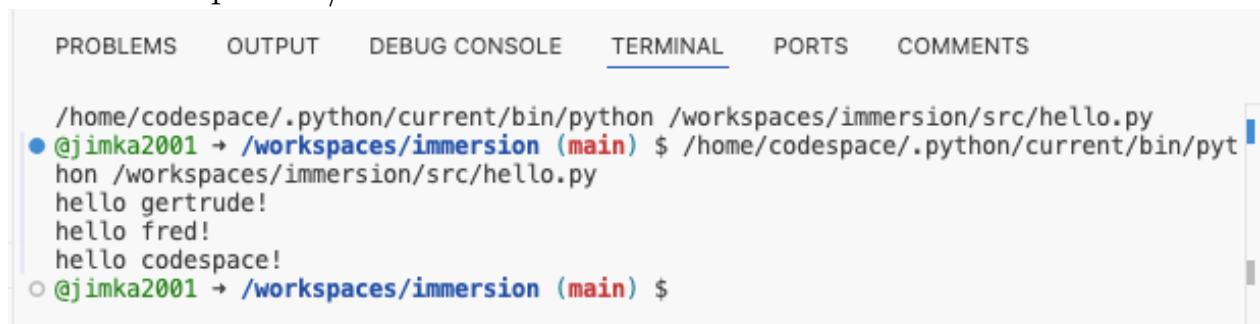
```
1 def hello(name):  
2     print("hello " + name + "!")
```

**Listing 3.2** (*Calls to Function `hello`*)

```
1 if __name__ == '__main__':  
2     # call the function with an argument  
3     hello("gertrude")  
4  
5     # second test  
6     hello("fred")
```

### 3.2 Make a Sample Run

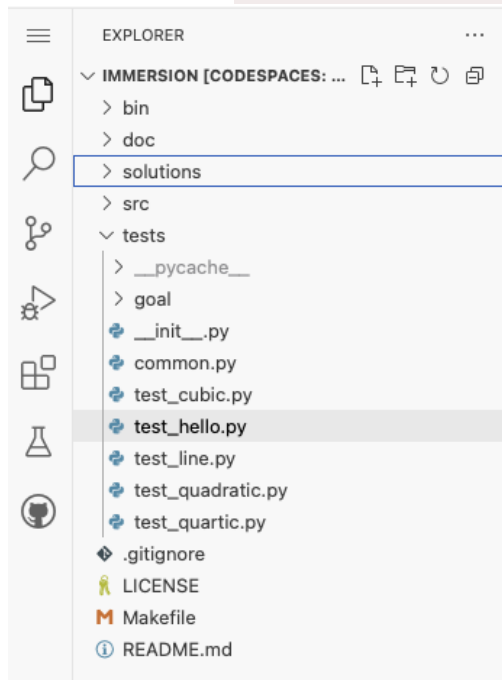
Find the icon  in the top-right of the editor window. Click the triangle, to see a sample run/execution of the code.



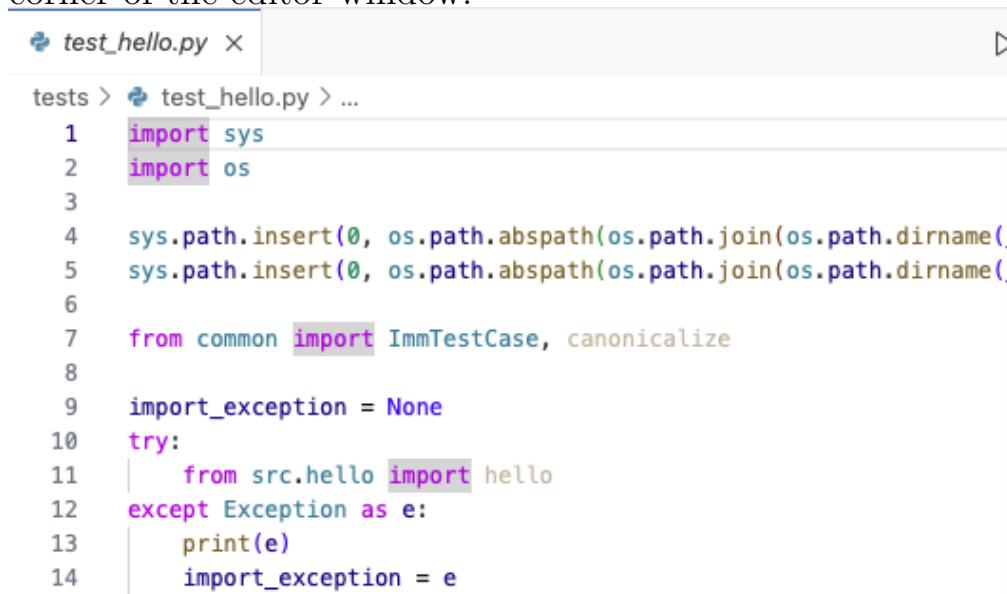
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS  
  
/home/codespace/.python/current/bin/python /workspaces/immersion/src/hello.py  
● @jimka2001 → /workspaces/immersion (main) $ /home/codespace/.python/current/bin/pyt  
hon /workspaces/immersion/src/hello.py  
hello gertrude!  
hello fred!  
hello codespace!  
○ @jimka2001 → /workspaces/immersion (main) $
```

### 3.3 Running Predefined Tests

Open the file `tests/test_hello.py` in a GitHub Code-Space.



To run the tests, press the  which you should find in the upper-right corner of the editor window.



The text window at the bottom of the editor should show the results of how many tests ran and whether there are any failures.

```
• @jimka2001 → /workspaces/immersion (main) $ /home/codespace/.python/current/bin/python /  
workspaces/immersion/tests/test_hello.py  
.hello test!  
.  
-----  
Ran 2 tests in 0.000s  
OK
```

### 3.4 Challenges for the Student

Understanding errors and debugging is difficult, but it is part of programming.

Experiment with the simple pieces of code in the above sections. Insert spaces and press the run button. Look at the error messages produced. Remove some quotation marks or parentheses (leaving unbalanced quotations marks or parentheses)—again look at what error messages you see when you try to run invalid code.

1. Remove and add some spaces at the beginning of a line.
2. Change the indentation.
3. Unbalance the parentheses.
4. Unbalance the quotation marks.
5. Put extra spaces inside the quotation marks.
6. Change the name of the `hello` function at definition site or call site.
7. Figure out how to undo your changes in the editor to make the code work again.
8. Run the tests `test/test_hello.py` as explained in Section 3.3.

## 4 Review of Polynomials (Optional)

Your goal in the next several sections is to find *root* of *polynomials*. But first, what is a polynomial? And then, what is a root?

### Definition 4.1 (*polynomial of degree $n$* )

A *polynomial* of *degree  $n$*  is a function of a single variable, usually  $x$ , of the form.

$$P(x) = \alpha_n x^n + \alpha_{n-1} x^{n-1} + \cdots + \alpha_2 x^2 + \alpha_1 x + \alpha_0$$

where  $\alpha_n, \alpha_{n-1} x^{n-1}, \dots, \alpha_0$  are called coefficients and may be integers, rational numbers, or real numbers.

An example of a 4th degree polynomial is

$$P(x) = 2x^4 + 5x^3 + x^2 - x + 1.$$

Any of the coefficients after the leading one may be 0, in which case the term is generally omitted. An example of a 3th degree polynomial where the coefficients of the  $x^2$  term is 0 is as follows.

$$P(x) = 5x^3 + x - 1.$$

### Definition 4.2 (*root*)

If  $P(x)$  is a polynomial, and  $r$  is a number for which  $P(r) = 0$ , then  $r$  is called a *root* of the polynomial.

The polynomial  $P(x) = x^3 - x$  has three roots, 1, -1, and 0. We know this because

$$P(1) = (1)^3 - 1 = 1 - 1 = 0$$

$$P(-1) = (-1)^3 - (-1) = -1 + 1 = 0$$

$$P(0) = (0)^3 - 0 = 0 - 0 = 0$$

### Theorem 4.3 (*Fundamental Theorem of Algebra*)

Every non-constant polynomial of degree 1 or higher has a root.

The Fundamental Theorem of Algebra claims that every polynomial. However, sometimes the root is complex. For example the polynomial,  $P(x) = x^2 + 1$ , has two complex roots. In this atelier, we only attempt to determine

the real roots, ignoring the complex roots. For this reason, we will not be able to find all the roots of some polynomials.

**Theorem 4.4 (*Roots with Multiplicity*)**

A polynomial of degree  $n \geq 1$  has  $n$  roots, some of which may be equal.

*Proof.* [By Induction]

- **Base case:** If  $n = 1$ , then Theorem 4.3 guarantees it has a root; call it  $r_1$ . Thus it is of the form  $P(x) = \alpha(x - r_1)$ .
- **Inductive case:** Suppose  $n > 1$ . Suppose every polynomial,  $Q(x)$  of degree  $n$  has  $n$ -many roots,  $r_1, \dots, r_n$ , and can thus be written as:

$$Q(x) = \alpha_1(x - r_1)(x - r_2) \cdots (x - r_n)$$

Now consider a polynomial,  $P(x)$ , of degree  $n + 1$ . Theorem 4.3 guarantees  $P(x)$  has a root; call it  $r_{n+1}$ , thus  $(x - r_{n+1})$  is a factor of  $P(x)$  so there exists a polynomial,  $Q(x)$ , of degree  $n$  such that  $P(x) = Q(x)(x - r_{n+1})$ . But by inductive hypothesis,  $Q(x)$  can be written as

$$Q(x) = \alpha_1(x - r_1)(x - r_2) \cdots (x - r_n)$$

So,

$$P(x) = \alpha_1(x - r_1)(x - r_2) \cdots (x - r_n)(x - r_{n+1})$$

Thus  $P(x)$  has  $n + 1$  roots.

□

Theorem 4.4 can be seen as a model of how we will find roots in this attempt. Given a degree  $n$  polynomial (for  $2 < n \leq 5$ ), if we successfully find a root,  $r_n$ , then we will factor  $(x - r_n)$  out of the polynomial, giving us new polynomial but with degree  $n - 1$ , which we can solve by the same approach. This process continues until we reach degree  $n = 2$  which we solve using the quadratic formula.



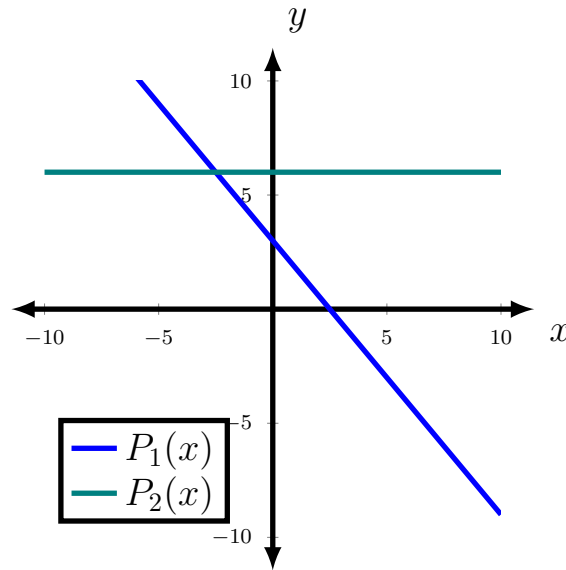


Figure 2: Line:  $y = P(x) = -1.2 * x + 3$

## 5 The Line: degree=1

### 5.1 Objectives

The assignment in this section is:

1. Complete the file `src/line.py` by writing a Python function which computes and returns the x-intercept of a line whose equation is  $y = ax + b$ .
2. You'll need to replace all occurrences of `raise NotImplementedError()` with correct python code which passes the tests.
3. Test the function in GitHub Code-Spaces by running the file `test/test_line.py`.

### 5.2 Overview

A polynomial of degree 1 is a function  $P(x) = ax + b$  whose graph is a line. If  $a = 0$  the line is horizontal and crosses the y-axis at  $y = b$ ; thus if  $b \neq 0$  then  $P(x) \neq 0$ . So if  $a = 0$  we will assume that  $P(x)$  has no root.

### 5.3 The Math / The Theoretical

Figure 2 shows the graphs of two polynomials of degree 1.

$$P_1(x) = -1.2x + 3$$

$$P_2(x) = 0x + 6$$

The line representing  $P_1(x)$  has an x-intercept computed as in Example 5.1.

**Example 5.1** (*Computing x-intercept of a line*)

$$P_1(x) = ax + b$$

$$0 = -1.2x + 3$$

$$x = \frac{3}{1.2} = 2.5$$

We see in Figure 2 that if the coefficient of  $x$  is 0, then the line is horizontal and has no x-intercept.  $P_2(x)$  is an example where  $a = 0$ . If the horizontal line  $P_2(x)$  is coincident with the x-axis, *i.e.*, if  $a = 0$  and  $b = 0$ , then there is no unique x-intercept.

If  $a \neq 0$ , then we have a line as shown in Figure 2. We can solve for the x-intercept by setting  $y$  to 0 and solving for  $x$ .

$$y = ax + b$$

$$0 = ax + b$$

$$-b = ax$$

$$\frac{-b}{a} = x$$

## 5.4 The Programming / The Practical

**Listing 5.2** (*Function declaration to find x-intercept of a line.*)

```

1 def find_x_intercept(a,b):
2     if a == 0:
3         # CHALLENGE: student must complete the implementation.
4         raise NotImplementedError()
5
6     else:
7         # CHALLENGE: student must complete the implementation.
8         raise NotImplementedError()

```

1. Open the file `src/line.py` in your web-based editor (GitHub CodeSpaces). You will develop the code in this file.

2. Find the occurrences of `raise NotImplementedError()`. This is the code that you must replace with working Python code.
3. You should update the code in the file `src/line.py` so that if  $a$  is 0, then the function returns an empty list and otherwise returns a list of the value  $\frac{-b}{a}$ .
4. You must figure out how to write that in the Python language. *I.e.*, you must figure out how to divide two numbers in Python and how to create a list, empty and otherwise.
5. Test your code by running the pre-defined tests in `tests/test_line.py`.
6. Look at the proposed solution in `solutions/line.py`. It is not necessary that your code match exactly.

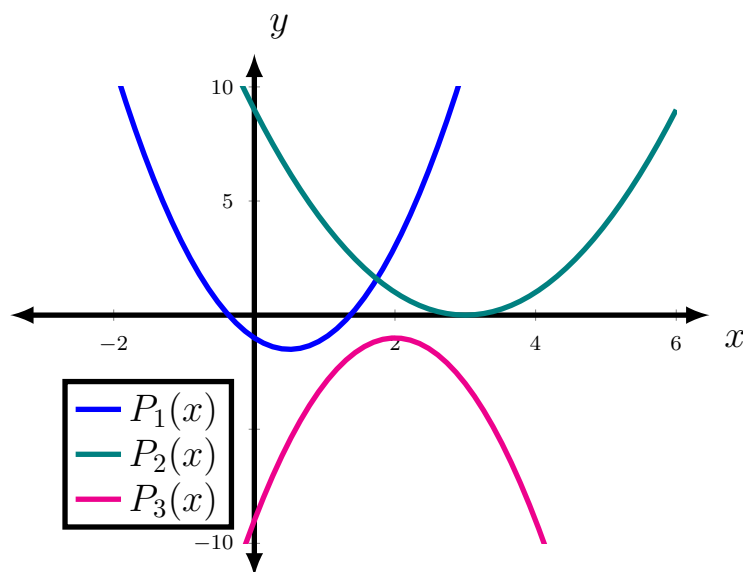


Figure 3: Parabolas

## 6 Quadratic: degree=2

### 6.1 Objectives

The assignment in this section is:

1. Complete the file `src/quadratic.py` by writing a Python function which computes and returns the x-intercepts of a parabola whose equation is

$$y = ax^2 + bx + c.$$

2. You'll need to replace all occurrences of `raise NotImplementedError()` with correct python code which passes the tests.
3. Test the function in GitHub Code-Spaces by running the file `test/test_quadratic.py`.

### 6.2 Overview

A quadratic polynomial is of the form  $P(x) = ax^2 + bx + c$ . If  $a = 0$ , then  $P(x)$  is actually a first degree polynomial which can be solved using the techniques described in Section 5; otherwise we need a more elaborate technique which is explained here in Section 6

We have the plots of three quadratic polynomials,  $P_1(x)$ ,  $P_2(x)$ , and  $P_3(x)$ , shown in Figure 3. Each plot takes the geometric form of a parabola. As can

be seen in the figure, the parabola might intersect the x-axis twice (*e.g.*,  $P_1(x)$ ), once (*e.g.*,  $P_2(x)$ ), or not at all (*e.g.*,  $P_3(x)$ ).

$$P_1(x) = 2x^2 - 2x - 1$$

$$P_2(x) = x^2 - x + \frac{1}{4}$$

$$P_3(x) = -2x^2 + 2x - 2$$

## 6.3 The Math / The Theoretical

To determine how many times a parabola touches the x-axis, we need to find the roots of the corresponding quadratic (degree 2) polynomial of the form  $ax^2 + bx + c$ . The formula is called the *quadratic formula*:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

In particular, we need to look at the expression under the square root, called the discriminant:  $b^2 - 4ac$ .

- If the discriminant is positive, then the polynomial has two distinct real roots.
- If the discriminant is zero, then the positive and negative square roots  $\pm\sqrt{b^2 - 4ac}$  in the quadratic formula are both 0; thus the polynomial has 1 single root where the parabola touches the x-axes tangentially.
- If the discriminant is negative, then there is a negative under the radical sign; thus the polynomial has no real roots. The polynomial does have complex roots; however we will ignore complex roots for this atelier.

## 6.4 The Programming / The Practical

Steps for computing roots of a polynomial of degree 2.

1. Assume the coefficients of  $P(x)$  are in the variables `a`, `b`, and `c`.
2. If `a == 0`, then delegate to previous solution by calling the function `find_x_intercept` and returning its value. Note that `find_x_intercept` takes two arguments, which it calls `a` and `b`, but these are not `a` and `b` of the quadratic. In fact for `find_x_intercept`,

`a` is the coefficients of  $x$  and `b` is the bias term of  $ax + b$ . However, for the quadratic, `a` is the coefficient of  $x^2$ , and `b` is the code of  $x$  in  $ax^2 + bx + c$ .

3. Compute the discriminant, storing in the variable `discriminant`.
4. Test three conditions.
  - (a) If the discriminant is positive, then return a list of the two roots.
  - (b) If the discriminant is zero (or very close to zero), then return a list (of length two) of the same root twice.
  - (c) If the discriminant is negative (  $< -\varepsilon$  ), then return an empty list `[]`.
5. Test your code by running the pre-defined tests in `tests/test_quadratic.py`.
6. Look at the proposed solution in `solutions/quadratic.py`. It is not necessary that your code match exactly.

**Listing 6.1** (*Function to compute roots of a quadratic polynomial.*)

```
1 def find_quadratic_roots(a, b, c):
2     epsilon = 0.001
3     discriminant = b * b - 4 * a * c
4     if a == 0:
5         # CHALLENGE: student must complete the implementation.
6         raise NotImplementedError()
7
8     if abs(discriminant) < epsilon:
9         # CHALLENGE: student must complete the implementation.
10        raise NotImplementedError()
11
12    elif discriminant > 0:
13        # CHALLENGE: student must complete the implementation.
14        raise NotImplementedError()
15
16    else:
17        # CHALLENGE: student must complete the implementation.
18        raise NotImplementedError()
```

## 7 Cubic: degree=3

### 7.1 Objectives

The assignment in this section is:

1. Complete the file `src/cubic.py` by writing a Python function which computes and returns the x-intercepts of a curve equation is

$$y = ax^3 + bx^2 + cx + d.$$

2. You'll need to replace all occurrences of `raise NotImplementedError()` with correct python code which passes the tests.
3. Complete the file `src/search.py` by completing the function `search_root_right` which follows a similar pattern to that of `search_root_left` which you have all the code for.
4. Test the function in GitHub Code-Spaces by running the file `test/test_cubic.py`.

### 7.2 Overview

We wish to find the roots of the cubic polynomial given by  $P(x) = ax^3 + bx^2 + cx + d$ . We will do this by first finding (approximating) a root,  $r$ , using a technique called *binary search*. Knowing root  $r$ , we know that  $(x - r)$  is a factor of  $ax^3 + bx^2 + cx + d$ . Once  $(x - r)$  is factored out of  $ax^3 + bx^2 + cx + d$  what remains is a quadratic polynomial of the form  $Ax^2 + Bx + C$ . We can find the roots of the quadratic using the technique explained in Section 6, thus obtaining the roots of the cubic polynomial.

### 7.3 The Math / The Theoretical

A polynomial of degree 3 has the form  $P(x) = ax^3 + bx^2 + cx + d$ . If  $a = 0$  then  $P(x)$  is really a quadratic polynomial (degree 2) and can be solved using the techniques described in Section 6. Every polynomial of degree 3 with real coefficients (for which  $a \neq 0$ ) has at least one real root because

1. Either  $\lim_{x \rightarrow -\infty} P(x) = -\infty$  and  $\lim_{x \rightarrow \infty} P(x) = \infty$ ,
2. Or  $\lim_{x \rightarrow -\infty} P(x) = \infty$  and  $\lim_{x \rightarrow \infty} P(x) = -\infty$ .

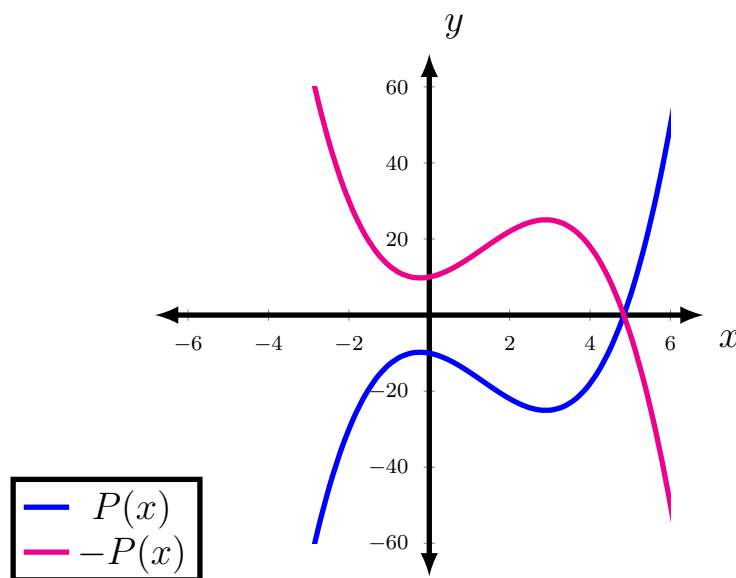


Figure 4: Cubics

We have a two cubic equations plotted in Figure 4.

$$P(x) = x^3 - 4 * x^2 - 2 * x - 10$$

$$-P(x) = -x^3 + 4 * x^2 + 2 * x + 10$$

$P(x)$  exemplifies the *standard* case where the leading coefficient is positive:  $a > 0$ .  $-P(x)$  exemplifies the alternate case where the leading coefficient is negative:  $a < 0$ . However, we notice that  $P(x)$  and  $-P(x)$  have the exact same roots, because if  $-P(x) = 0$ , then  $P(x) = 0$ . Therefore, if  $a < 0$ , we can simply find the roots of  $-ax^3 - bx^2 - cx - d$ ; *i.e.*, we simply negate the coefficients and find the roots of the negated cubic polynomial.

Given a cubic polynomial and a root,  $r$ , we may factor the polynomial into the product of a monomial  $(x - r)$  and a quadratic polynomial.

If  $P(x) = ax^3 + bx^2 + cx + d$  has a root at  $x = r$ , then  $P(r) = 0$ . Consequently,  $(x - r)$  is a factor of  $ax^3 + bx^2 + cx + d$ , and  $P(x) = (x - r)(Ax^2 + Bx + C)$  for some  $A, B, C$ .

$$P(x) = (ax^3 + bx^2 + cx + d)$$

$$= (x - r)(Ax^2 + Bx + C) \tag{1}$$

Once  $A$ ,  $B$ , and  $C$  have been found, then the two remaining roots can be found by applying the quadratic formula to  $Ax^2 + Bx + C$ . So how can



we determine the values of  $A$ ,  $B$ , and  $C$  given the root  $r$  and the original coefficients  $a$ ,  $b$ ,  $c$ , and  $d$ ?

We can verify that  $A$ ,  $B$ , and  $C$  are determined by the following equations.

$$A = a \tag{2}$$

$$\begin{aligned} B &= b + ar \\ &= b + Ar \end{aligned} \tag{3}$$

$$\begin{aligned} C &= c + br + ar^2 \\ &= c + (b + ar)r \\ &= c + Br \end{aligned} \tag{4}$$

The equations for  $A$ ,  $B$ , and  $C$  follow a regular and predictable pattern which may not be immediately apparent. However, the pattern may become more clear when looking at the analogous derivation for the quartic (Equations (5) through (8)) page 31, and also for the quintic (Equations (9) through (13)) page 34.

We may verify this factorization by substituting Equations (2), (3), and (4) into  $(x - r)(Ax^2 + Bx + C)$ , then working through some tedious algebra.

$$\begin{aligned} (x - r)(Ax^2 + Bx + C) &= (x - r)(ax^2 + (b + ar)x + (c + br + ar^2)) \\ &= (x - r)(ax^2 + bx + arx + c + br + ar^2) \\ &= x(ax^2 + bx + arx + c + br + ar^2) \\ &\quad - r(ax^2 + bx + arx + c + br + ar^2) \\ &= ax^3 + bx^2 + arx^2 + cx + brx + ar^2x \\ &\quad - arx^2 - brx - ar^2x - cr - br^2 - ar^3 \\ &= ax^3 + bx^2 + cx \\ &\quad + \underbrace{arx^2 + brx + ar^2x - (arx^2 + brx + ar^2x)}_{=0} \\ &\quad - cr + br^2 - ar^3 \\ &= ax^3 + bx^2 + cx + \underbrace{(d - d)}_{+0} - cr - br^2 - ar^3 \\ &= \underbrace{ax^3 + bx^2 + cx + d}_{=P(x)} - \underbrace{(d + cr + br^2 + ar^3)}_{=P(r)} \\ &= P(x) - P(r) \\ &= P(x) \end{aligned}$$

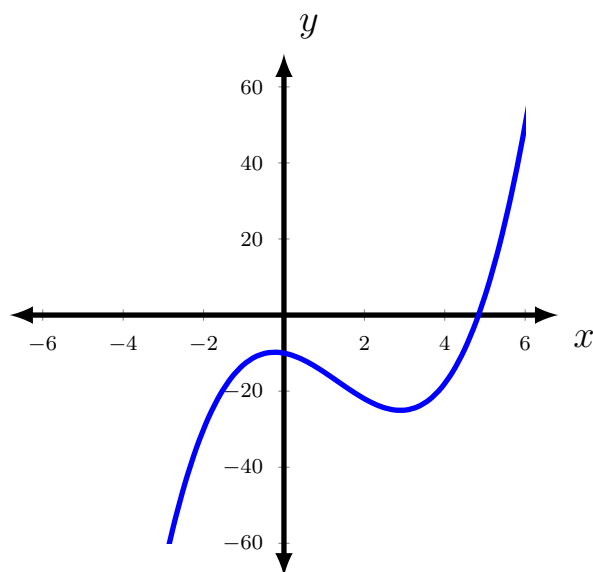


Figure 5: Expanding Binary Search

Since  $r$  is assumed to be a root of  $P(x)$ , then we know that  $P(r) = 0$ . Thus  $P(x) - P(r) = P(x)$ .

## 7.4 The Programming / The Practical

This challenge contains two parts:

- In **Find a root using Binary Search**, Section 7.5, you will complete the file `src/search.py`.
- In **Cubic Roots**, Section 7.6, you will complete the file `src/cubic.py`.

## 7.5 Find a root using Binary Search

In this section you will complete the code in `src/search.py`. This code will be reused in Section 7.6 and again in Section 8. The purpose of the code is to find one root (any root) of a given polynomial. The purposes of finding one root is so we can factor  $(x - r)$  out of the polynomial, thus reducing a degree 3 polynomial to a degree 2, or in general reducing a degree  $n$  to a degree  $n - 1$  polynomial.

Consider the graph of the cubic polynomial shown in Figure 5. We'd like to iteratively find a root; we will use the following steps.

1. If  $P(0) = 0$ , then we know the root,  $x = 0$ .

2. If  $P(0) < 0$ , then find a value of  $x_{upper}$  (to the right of 0,  $x_{upper} > 0$ ) such that either  $P(x_{upper}) > 0$ . Now take  $x_{lower} = 0$ . Thus we will know there is a root in the interval  $[x_{lower}, x_{upper}]$ .
3. If  $P(0) > 0$ , then find a value of  $x_{lower}$  (to the left of 0,  $x_{lower} < 0$ ) such that either  $P(x_{lower}) < 0$ . Now take  $x_{upper} = 0$ . Thus we will know there is a root in the interval  $[x_{lower}, x_{upper}]$ .
4. Since we know there is a root in the interval  $[x_{lower}, x_{upper}]$ , we can divide the interval into two intervals. With the midpoint of the interval,

$$x_{mid} = \frac{x_{upper} + x_{lower}}{2},$$

we consider two intervals:  $[x_{lower}, x_{mid}]$  and  $[x_{mid}, x_{upper}]$ . At least one of the following is true:

- (a)  $x_{upper} - x_{right} < \varepsilon$ , for some small epsilon (*e.g.*,  $\varepsilon = 0.00001$ ), then we know the root is  $x_{mid} \pm \varepsilon$ , which is good enough.
- (b)  $P(x_{mid}) = 0$ , then we know the root  $x = x_{mid}$ .
- (c) There is a root in the interval  $[x_{lower}, x_{mid}]$ , then we repeat step 4 on the interval  $[x_{lower}, x_{mid}]$ .
- (d) There is a root in the interval  $[x_{mid}, x_{upper}]$ , then we repeat step 4 on the interval  $[x_{mid}, x_{upper}]$ .

How does step 2 work? Start with  $x_{upper} = 1$ , and query whether  $P(x_{upper}) < 0$ . If so, then double  $x_{upper}$  and try again, until  $P(x_{upper}) > 0$ . As an example with the polynomial in Figure 5, we would try  $P(1) < 0$ ,  $P(2) < 0$ ,  $P(4) < 0$ , and finally  $P(8) > 0$ .

Similarly for step 3? Start with  $x_{upper} = -1$ , and query whether  $P(x_{lower}) > 0$ . If so, then double  $x_{upper}$  and try again, until  $P(x_{upper}) < 0$ .

## 7.6 Cubic Roots

Steps for computing roots of a cubic polynomial.

1. Assume the coefficients of  $P(x) = ax^3 + bx^2 + cx + d$  are **a**, **b**, **c**, and **d**.
2. If **a==0**, then delegate to the previous solution by calling **find\_quadratic\_roots** and returning its return value. Be careful, **find\_quadratic\_roots** accepts 3 input parameters.

3. If  $P(x)$  is of the form of  $-P(x)$  in Figure 4, *i.e.*, if  $a < 0$ , then compute and return the roots of  $-P(x)$ . To compute the roots of  $-P(x)$  we simply return `find_cubic_roots(-a, -b, -c, -d)`.
4. Since  $P(0) = d$ , then we can easily evaluate the polynomial at  $x = 0$  to get its y-intercept.
  - (a) If  $d = 0$ , then 0 is a root,  $P(0) = 0$ .
  - (b) If  $d > 0$  then there is a root on the negative x-axis. Find it with a binary search.
  - (c) If  $d < 0$  then there is a root on the positive x-axis. Find it with a binary search.
5. See Section 7.5 to explain the binary search.
6. Once the root  $r$  is found, this means  $(x - r)$  factors out  $P(x)$ . *I.e.*,  $P(x) = (x - r)(Ax^2 + Bx + C)$ . The formulas for  $A$ ,  $B$ , and  $C$  can be found in Equations (2), (3), and (4).
7. The roots of the cubic are `[r]` concatenated to the roots of the quadratic  $Ax^2 + Bx + C$ , which you can compute using the techniques in Section 6.
8. Test your code by running the pre-defined tests in `tests/test_cubic.py`.
9. Look at the proposed solution in `solutions/cubic.py`. It is not necessary that your code match exactly.

## 8 Quartic: degree=4

### 8.1 Objectives

The assignment in this section is:

1. Complete the file `src/quartic.py` by writing a Python function which computes and returns the x-intercepts of a curve equation is

$$y = ax^4 + bx^3 + cx^2 + dx + e.$$

2. You'll need to replace all occurrences of `raise NotImplementedError()` with correct python code which passes the tests.
3. Test the function in GitHub Code-Spaces by running the file `test/test_quartic.py`.

### 8.2 Overview

We wish to find the roots of the quartic polynomial given by

$$P(x) = ax^4 + bx^3 + cx^2 + dx + e.$$

We will do this by first finding (approximating) a root,  $r$ , using a technique called *binary search*. We will determine which regions to search by examining the derivative

$$\frac{d}{dx}P(x) = 4ax^3 + 3bx^2 + 2cx + d.$$

Knowing root  $r$ , we know that  $(x-r)$  is a factor of  $ax^4 + bx^3 + cx^2 + dx + e$ . Once  $(x-r)$  is factored out of  $P(x)$  what remains is a cubic polynomial of the form  $Ax^3 + Bx^2 + Cx + D$ . We can find the roots of the cubic using the technique explained in Section 7, thus obtaining the roots of the quartic polynomial.

### 8.3 The Math / The Theoretical

A polynomial of degree 4 has the form  $P(x) = ax^4 + bx^3 + cx^2 + dx + e$ . If  $a = 0$  then  $P(x)$  is really a cubic polynomial (degree 3) and can be solved using the techniques described in Section 7.

If  $a < 0$  then similar to what we did for the cubic polynomial in Section 7.3, we instead find the roots of  $-P(x) = -ax^4 - bx^3 - cx^2 - dx - e$ .  $P(x)$  and

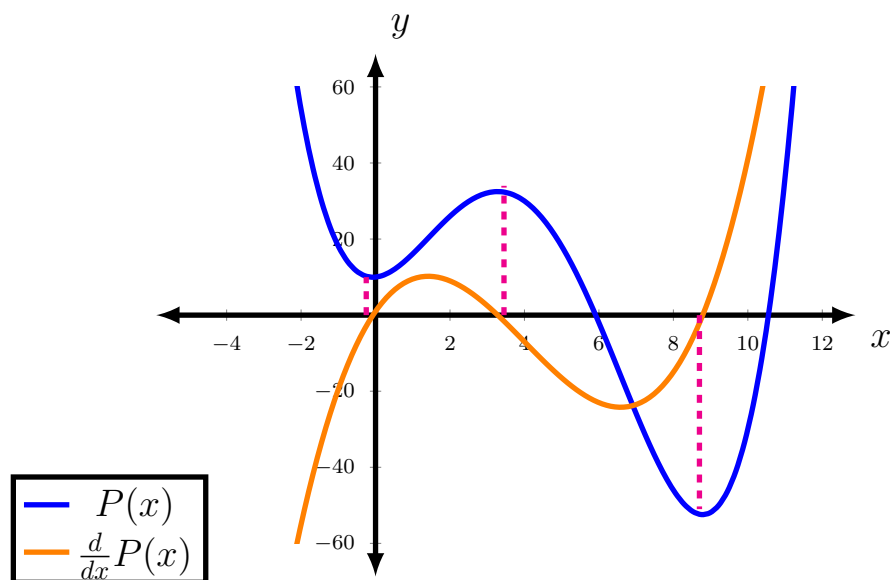


Figure 6: Quartics

$-P(x)$  are guaranteed to have the same roots, but the code in Python will be easier if we know that  $a \geq 0$ .

Two polynomial equations (for  $P(x)$  and  $\frac{d}{dx}P(x)$ ) are plotted in Figure 6.

$$P(x) = \frac{1}{8}x^4 - 2x^3 + 7x^2 + x + 6$$

$$\frac{d}{dx}P(x) = \frac{1}{2}x^3 - 6x^2 + 14x + 1$$

Notice that at the extreme left and right, the plot of  $P(x)$  opens upward, and the graph changes direction 3 times. The values of  $x$  at which  $P(x)$  changes direction are called the *inflection points*. The inflection points are local minima of  $P(x)$ , and can be computed by finding the values for which  $\frac{d}{dx}P(x) = 0$ . Since the derivative of a degree 4 polynomial is a degree 3 (cubic) polynomial, we can find where  $\frac{d}{dx}P(x) = 0$  by using the techniques explained in Section 7.

Once we have determined the 3 inflection points  $x_1$ ,  $x_2$ , and  $x_3$ , we can simply test  $P(x_1)$ ,  $P(x_2)$ ,  $P(x_3)$ . If one of these values is 0, then we have found a root,  $r$ , and we can factor  $(x - r)$  out of  $P(x)$  to obtain a cubic polynomial. We can find the roots of the cubic using the techniques explained in Section 7.

If none of  $P(x_1)$ ,  $P(x_2)$ , or  $P(x_3)$  is zero, then we check whether one of them is negative. As shown in Figure 6, whenever  $P(x) < 0$  then there is a root both to the right and left of  $x$ . If we can find that root, then as before we can factor out  $(x - r)$  to get a cubic polynomial, and then find its roots—

thus obtaining the roots of the quartic.

Once we have determined a root,  $r$  of  $P(x)$ , then we know (similar to the cubic case in Section 7.3) that  $P(x)$  factors into  $(x - r)(Ax^3 + Bx^2 + Cx + D)$ , where

$$A = a \tag{5}$$

$$\begin{aligned} B &= b + ar \\ &= b + Ar \end{aligned} \tag{6}$$

$$\begin{aligned} C &= c + br + ar^2 \\ &= c + (b + ar)r \\ &= c + Br \end{aligned} \tag{7}$$

$$\begin{aligned} D &= d + cr + br^2 + ar^3 \\ &= d + (c + br + ar^2)r \\ &= d + Cr \end{aligned} \tag{8}$$

## 8.4 The Programming / The Practical

Steps for finding the roots of a quartic polynomial.

1. Assume the coefficients of  $P(x) = ax^4 + bx^3 + cx^2 + dx + e$  are **a**, **b**, **c**, **d**, and **e**.
2. If **a==0**, then delegate to the previous solution by calling **find\_cubic\_roots** and returning its return value. Be careful, **find\_cubic\_roots** accepts 4 input parameters.
3. Since  $P(0) = e$ , then we can easily evaluate the polynomial at  $x = 0$  to get its y-intercept.
  - (a) If  $e = 0$ , then 0 is a root,  $P(0) = 0$ .
  - (b) If  $e < 0$  then there is a root on the positive x-axis and a root on the negative x-axis. Find it with a binary search.
4. Find the inflection points by finding the roots of

$$\frac{d}{dx}P(x) = 4ax^3 + 3bx^2 + 2cx + d.$$

- (a) If  $P(x)$  is zero at one of these inflection points, then the inflection point is a root.

- (b) If  $P(x) < 0$  at an inflection point, then there is a root both to the right and the left; find with by calling either `search_root_right` or `search_root_left`.
5. Once a root has been found, factor
- $$P(x) = (x - r)(Ax^3 + Bx^2 + Cx + D)$$
- by finding the coefficients,  $A$ ,  $B$ ,  $C$ , and  $D$ , using Equations (5) through (8).
6. Find the roots of  $Ax^3 + Bx^2 + Cx + D$  using `find_cubic_roots`.
7. Concatenate the list of all the roots together to obtain the roots of the quartic polynomial.
8. Test your code by running the pre-defined tests in `tests/test_quartic.py`.
9. Look at the proposed solution in `solutions/quartic.py`. It is not necessary that your code match exactly.



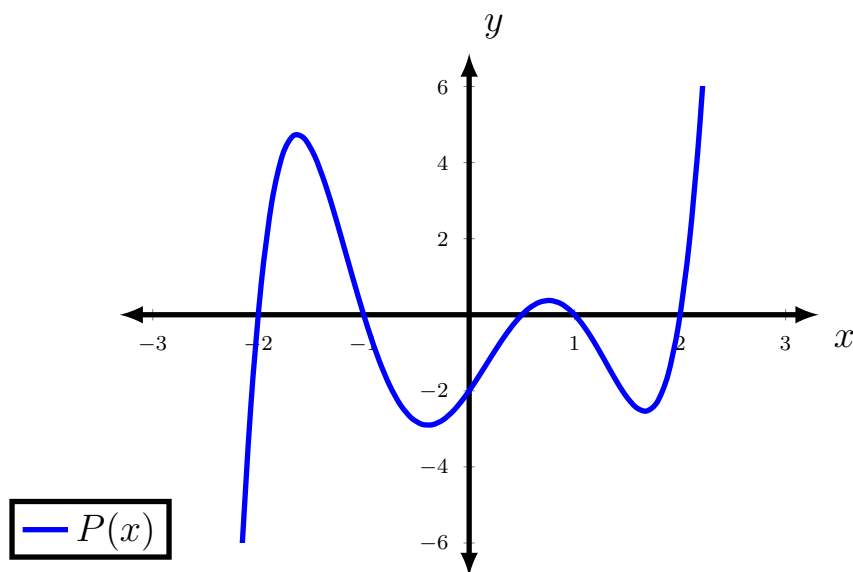


Figure 7: Quintic

## 9 Quintic: degree=5

This exercise is left as an exercise for the student. However, the following are some notes to help lead the student in the right direction.

$$P(x) = x^5 - \frac{1}{2}x^4 + 5x^3 + \frac{5}{2}x^2 + 4x - 2$$

A quintic polynomial has the form  $P(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$ . If  $a = 0$  then  $P(x)$  is really a quartic polynomial (degree 4) and can be solved using the techniques described in Section 8. Since a quintic polynomial has odd degree (if  $a \neq 0$ ), then, like the cubic (Section 7), it is guaranteed to have a real root. This fact is guaranteed, because if  $a > 0$  then  $P(x) \rightarrow \infty$  for  $x \gg 0$  and  $P(x) \rightarrow -\infty$  for  $x \ll 0$ .

Given this fact, we can find a root,  $r$  using a binary search as we did for the cubic. Since  $P(0) = f$ , then we can consider two cases.

1. If  $f > 0$  then search for a root on the negative x-axis
2. If  $f < 0$  then search for a root on the positive x-axis

Once the root  $r$  is found, factor out  $(x-r)$  from  $P(x)$  to result in a quartic, which we can solve using the technique in Section 8.

$$\begin{aligned} P(x) &= ax^5 + bx^4 + cx^3 + dx^2 + ex + f \\ &= (x - r)(Ax^4 + Bx^3 + Cx^2 + Dx + E) \end{aligned}$$

Where

$$A = a \tag{9}$$

$$\begin{aligned} B &= b + ar \\ &= b + Ar \end{aligned} \tag{10}$$

$$\begin{aligned} C &= c + br + ar^2 \\ &= c + (b + ar)r \\ &= c + Br \end{aligned} \tag{11}$$

$$\begin{aligned} D &= d + cr + br^2 + ar^3 \\ &= d + (c + br + ar^2)r \\ &= d + Cr \end{aligned} \tag{12}$$

$$\begin{aligned} E &= e + dr + cr^2 + br^3 + ar^4 \\ &= e + (d + cr + br^2 + ar^3)r \\ &= e + Dr \end{aligned} \tag{13}$$

Having determined  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $E$ , the roots of

$$Ax^4 + Bx^3 + Cx^2 + Dx + E$$

can be found with a call to `find_quartic_roots`, which you implemented in Section 8.