Hints

Why doesn't implemented python script do anything if you run it?

```
python main.py
```

```
poetry run cliname --option arg
```

Square Roots

- Implement two different square root algorithms
  - exhaustive
  - bisection search
- Fill out the CLI function `squareroot`
- Use a profiler!

pyinstrument

Project description

pyinstrument

Pyinstrument is a Python profiler. A profiler is a tool to help you 'optimize' your code - make it faster. It sounds obvious, but to get the biggest speed increase you must focus on the slowest part of your program. Pyinstrument helps you find it!

Can also provide summary about total time taken

pyinstrument

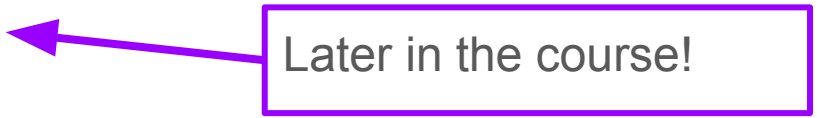
```
from pyinstrument import Profiler

profiler = Profiler()
profiler.start()

# code you want to profile

profiler.stop()
```

These ^^^ aspects will be located throughout your code.

Which algorithms is faster?

Measure the time it takes

Analyze the algorithm for "complexity"



Proactive Programmers

https://proactiveprogrammers.com/data-abstraction/programming-projects/square-root/

READ THIS! IT WILL HELP!

Previous Notes From Class

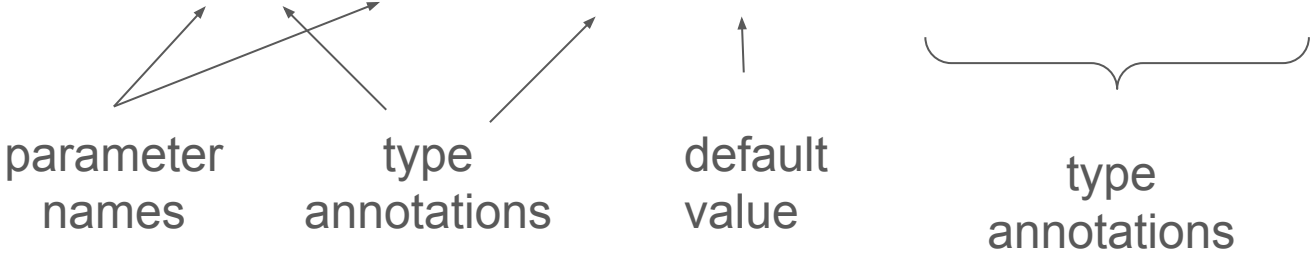
https://colab.research.google.com/github/allegheny-college-cmpsc-101-spring-2024/course-materials/blob/main/notes/20240129_numerical_computation.ipynb

Guttag Chapter 3

Additional notes from proactive programmers:
https://githubtocolab.com/ProactiveProgrammers/www.proactiveprogrammers.com/blob/master/files/data-abstraction/numerical-computation/calculate-approximate-square-root.ipynb

Closer look at type annotations and default values

```
def compute_square_root_exhaustive(x: int, epsilon: float = 0.01) -> Tuple[bool, float,
int]:
```



close the approximation of `x`'s square root must be. The notation `Tuple[bool, float, int]` that describes the output of these functions shows that they each return three values. The first variable in the return value is a `bool` indicating whether or not the function found an answer within the tolerance of `epsilon`. Finally, the second returned variable is a `float` for the calculated value of the square root and the third one is an `int` for the number of guesses that the algorithm took.

Prime Testing

Last engineering effort before the midterm!

You will implement two functions to determine if a number is prime

Again, using profiler to show which function is faster

Additional computation by hand to find difference in algorithm efficiency

Notes

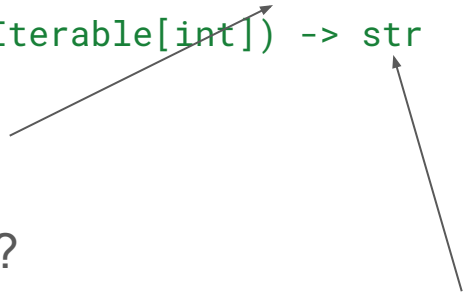
https://proactiveprogrammers.com/data-abstraction/engineering-efforts/primality-testing/

https://githubtocolab.com/ProactiveProgrammers/www.proactiveprogrammers.com/blob/master/files/data-abstraction/numerical-computation/perform-primality-testing.ipynb

Class notes coming on Friday

Primes have no factors (other than 1 and themselves)

- `def human_readable_boolean(answer: bool) -> str`
- `def pretty_print_list(values: Iterable[int]) -> str`



Is 16 prime? False || Is 16 prime? No!

- which version is human readable?

The factors of 16 are [1 2 4 8] || The factors of 16 are 1, 2, 4, 8

- which version is "prettier"?

writing import statements and creating CLI and profile objects

reference you earlier projects! for example, square roots in main.py has:

```
2
3   from pyinstrument import Profiler  # type: ignore
4
5   from typing import Tuple
6
7   from enum import Enum
8
9   import typer
10
11  from rich.console import Console
12
13  # create a Typer object to support the command-line interface
14  cli = typer.Typer()
15
16  # create a Profiler object to support timing program code segments
17  profiler = Profiler()
18
```

Search discord - top right corner