CS 115 Lecture 5

Math library; building a project

Neil Moore

Department of Computer Science
University of Kentucky
Lexington, Kentucky 40506
neil@cs.uky.edu

15 September 2015

The `math` library

We've seen already how to do everything a five-function calculator can do. What about more advanced math?

The `math` library

We've seen already how to do everything a five-function calculator can do. What about more advanced math?

- That's available in Python too.
- But it's not built-in like + and `float` are.

The `math` library

We've seen already how to do everything a five-function calculator can do. What about more advanced math?

- That's available in Python too.
- But it's not built-in like + and `float` are.
- Instead it's in a **library**.
  - A collection of pre-written code intended to be re-used.

The `math` library

We've seen already how to do everything a five-function calculator can do. What about more advanced math?

- That's available in Python too.
- But it's not built-in like + and `float` are.
- Instead it's in a **library**.
  - A collection of pre-written code intended to be re-used.
    - ★ Functions
    - ★ Constants
    - ★ Types ("classes")

The `math` library

We've seen already how to do everything a five-function calculator can do. What about more advanced math?

- That's available in Python too.
- But it's not built-in like + and `float` are.
- Instead it's in a **library**.
  - A collection of pre-written code intended to be re-used.
    - ★ Functions
    - ★ Constants
    - ★ Types ("classes")
  - The `math` library comes with Python.
  - `graphics` (chapter 3) is a **third-party** library.

The `math` library

We've seen already how to do everything a five-function calculator can do. What about more advanced math?

- That's available in Python too.
- But it's not built-in like + and `float` are.
- Instead it's in a **library**.
  - A collection of pre-written code intended to be re-used.
    - ★ Functions
    - ★ Constants
    - ★ Types ("classes")
  - The `math` library comes with Python.
  - `graphics` (chapter 3) is a **third-party** library.
- The Python `math` library has:
  - Functions for trig, logarithms, and more.
  - Constants for π and e.

The `math` library

We've seen already how to do everything a five-function calculator can do. What about more advanced math?

- That's available in Python too.
- But it's not built-in like + and `float` are.
- Instead it's in a **library**.
  - A collection of pre-written code intended to be re-used.
    - ★ Functions
    - ★ Constants
    - ★ Types ("classes")
  - The `math` library comes with Python.
  - `graphics` (chapter 3) is a **third-party** library.
- The Python `math` library has:
  - Functions for trig, logarithms, and more.
  - Constants for π and e.

Using libraries in Python

- To use a library, you **import it**:

```
import math
```

Using libraries in Python

- To use a library, you **import it**:

```
import math
```

  - Put this at the very top of the program.
  - After headers comments, before "`def main():`"

Using libraries in Python

- To use a library, you **import it**:

  ```
  import math
  ```

  - Put this at the very top of the program.
  - After headers comments, before "`def main():`"
- Then your program can use the functions in the library.
  - Their names are *`library.something`*
  - So `math.log` (function), `math.pi` (constant)

Using libraries in Python

- To use a library, you **import it**:

  ```
  import math
  ```

  - Put this at the very top of the program.
  - After headers comments, before "`def main():`"
- Then your program can use the functions in the library.
  - Their names are *`library.something`*
  - So `math.log` (function), `math.pi` (constant)
  - Call functions with parenthesized arguments afterwards
    - ★ Just like `input` and `print`
  - Each function has its own rules about what arguments it accepts.

Using libraries in Python

- To use a library, you **import it**:

```
import math
```

  - Put this at the very top of the program.
  - After headers comments, before "`def main():`"
- Then your program can use the functions in the library.
  - Their names are *`library.something`*
  - So `math.log` (function), `math.pi` (constant)
  - Call functions with parenthesized arguments afterwards
    - ★ Just like `input` and `print`
  - Each function has its own rules about what arguments it accepts.
  - If the function returns a value, use it as an expression:

```
height = math.log(size, 2)
```

  - If it does not, use it as an entire statement:

```
random.seed()
```

Using libraries in Python

- To use a library, you **import it**:

```
import math
```

  - Put this at the very top of the program.
  - After headers comments, before "`def main():`"
- Then your program can use the functions in the library.
  - Their names are *`library.something`*
  - So `math.log` (function), `math.pi` (constant)
  - Call functions with parenthesized arguments afterwards
    - ★ Just like `input` and `print`
  - Each function has its own rules about what arguments it accepts.
  - If the function returns a value, use it as an expression:

```
height = math.log(size, 2)
```

  - If it does not, use it as an entire statement:

```
random.seed()
```

- Only import the libraries you need!

Using libraries in Python

- To use a library, you **import it**:

```
import math
```

  - Put this at the very top of the program.
  - After headers comments, before "`def main():`"
- Then your program can use the functions in the library.
  - Their names are *`library.something`*
  - So `math.log` (function), `math.pi` (constant)
  - Call functions with parenthesized arguments afterwards
    - ★ Just like `input` and `print`
  - Each function has its own rules about what arguments it accepts.
  - If the function returns a value, use it as an expression:

```
height = math.log(size, 2)
```

  - If it does not, use it as an entire statement:

```
random.seed()
```

- Only import the libraries you need!

Save yourself typing

- You can instead import particular functions or constants:

```
from math import sin, cos, tan
```

  - List the names you are importing, comma-separated.
  - Then you can use them without the "`math.`"

```
y = sin(angle) * radius
```

Save yourself typing

- You can instead import particular functions or constants:

```
from math import sin, cos, tan
```

  - List the names you are importing, comma-separated.
  - Then you can use them without the "`math.`"

```
y = sin(angle) * radius
```

  - Saves typing if you're calling a few functions many times.

Save yourself typing

- You can instead import particular functions or constants:

```
from math import sin, cos, tan
```

  - List the names you are importing, comma-separated.
  - Then you can use them without the "`math.`"

```
y = sin(angle) * radius
```

  - Saves typing if you're calling a few functions many times.

- One last way:

```
from math import *
```

Save yourself typing

- You can instead import particular functions or constants:

  ```
  from math import sin, cos, tan
  ```

  - List the names you are importing, comma-separated.
  - Then you can use them without the "`math.`"

    ```
    y = sin(angle) * radius
    ```

  - Saves typing if you're calling a few functions many times.
- One last way:

  ```
  from math import *
  ```

  - Imports everything in the library.
  - And you don't have to use "`math.`"

    ```
    num = e ** pi
    ```

Save yourself typing

- You can instead import particular functions or constants:

  ```
  from math import sin, cos, tan
  ```

  - List the names you are importing, comma-separated.
  - Then you can use them without the "`math.`"

    ```
    y = sin(angle) * radius
    ```

  - Saves typing if you're calling a few functions many times.
- One last way:

  ```
  from math import *
  ```

  - Imports everything in the library.
  - And you don't have to use "`math.`"

    ```
    num = e ** pi
    ```

  - Sounds great, right?

Save yourself typing

- You can instead import particular functions or constants:

  ```
  from math import sin, cos, tan
  ```

  - List the names you are importing, comma-separated.
  - Then you can use them without the "`math.`"

    ```
    y = sin(angle) * radius
    ```

  - Saves typing if you're calling a few functions many times.
- One last way:

  ```
  from math import *
  ```

  - Imports everything in the library.
  - And you don't have to use "`math.`"

    ```
    num = e ** pi
    ```

  - Sounds great, right? There's a catch...

Save yourself typing

- You can instead import particular functions or constants:

  ```
  from math import sin, cos, tan
  ```

  - List the names you are importing, comma-separated.
  - Then you can use them without the "`math.`"

    ```
    y = sin(angle) * radius
    ```

  - Saves typing if you're calling a few functions many times.
- One last way:

  ```
  from math import *
  ```

  - Imports everything in the library.
  - And you don't have to use "`math.`"

    ```
    num = e ** pi
    ```

  - Sounds great, right? There's a catch...
    - ★ What if Python 3.5 adds a new math function?

Save yourself typing

- You can instead import particular functions or constants:

  ```
  from math import sin, cos, tan
  ```

  - List the names you are importing, comma-separated.
  - Then you can use them without the "`math.`"

    ```
    y = sin(angle) * radius
    ```

  - Saves typing if you're calling a few functions many times.
- One last way:

  ```
  from math import *
  ```

  - Imports everything in the library.
  - And you don't have to use "`math.`"

    ```
    num = e ** pi
    ```

  - Sounds great, right? There's a catch...
    - ★ What if Python 3.5 adds a new math function?
    - ★ And you already had a variable with that name...
    - ★ Your code could break!

Save yourself typing

- You can instead import particular functions or constants:

  ```
  from math import sin, cos, tan
  ```

  - List the names you are importing, comma-separated.
  - Then you can use them without the "`math.`"

    ```
    y = sin(angle) * radius
    ```

  - Saves typing if you're calling a few functions many times.
- One last way:

  ```
  from math import *
  ```

  - Imports everything in the library.
  - And you don't have to use "`math.`"

    ```
    num = e ** pi
    ```

  - Sounds great, right? There's a catch...
    - ★ What if Python 3.5 adds a new math function?
    - ★ And you already had a variable with that name...
    - ★ Your code could break!
  - Professional Python programmers avoid `from lib import *`, but we'll use it occasionally in labs and homework.

Save yourself typing

- You can instead import particular functions or constants:

  ```
  from math import sin, cos, tan
  ```

  - List the names you are importing, comma-separated.
  - Then you can use them without the "`math.`"

    ```
    y = sin(angle) * radius
    ```

  - Saves typing if you're calling a few functions many times.
- One last way:

  ```
  from math import *
  ```

  - Imports everything in the library.
  - And you don't have to use "`math.`"

    ```
    num = e ** pi
    ```

  - Sounds great, right? There's a catch...
    - ★ What if Python 3.5 adds a new math function?
    - ★ And you already had a variable with that name...
    - ★ Your code could break!
  - Professional Python programmers avoid `from lib import *`, but we'll use it occasionally in labs and homework.

What's in the `math` library

- Trigonometry: `sin`, `cos`, `tan`, `cosh`, ...

```
angle = math.atan(y/x)
circumference = math.pi * diameter
```

What's in the `math` library

- Trigonometry: `sin`, `cos`, `tan`, `cosh`, ...

```
angle = math.atan(y/x)
circumference = math.pi * diameter
```

- Natural log, and other bases:

```
doubling_time = math.log(2) / rate
pH = -log(activity, 10)
```

What's in the `math` library

- Trigonometry: `sin`, `cos`, `tan`, `cosh`, ...

```
angle = math.atan(y/x)
circumference = math.pi * diameter
```

- Natural log, and other bases:

```
doubling_time = math.log(2) / rate
pH = -log(activity, 10)
```

- e and e^x:

```
balance = principal * math.e ** (rate * time)
balance = principal * math.exp(rate * time)
```

What's in the `math` library

- Trigonometry: `sin`, `cos`, `tan`, `cosh`, ...

```
angle = math.atan(y/x)
circumference = math.pi * diameter
```

- Natural log, and other bases:

```
doubling_time = math.log(2) / rate
pH = -log(activity, 10)
```

- e and e^x:

```
balance = principal * math.e ** (rate * time)
balance = principal * math.exp(rate * time)
```

- More: `sqrt`, `factorial`, ...

```
https://docs.python.org/3/library/math.html
```

What's in the `math` library

- Trigonometry: `sin`, `cos`, `tan`, `cosh`, ...

```
angle = math.atan(y/x)
circumference = math.pi * diameter
```

- Natural log, and other bases:

```
doubling_time = math.log(2) / rate
pH = -log(activity, 10)
```

- e and e^x:

```
balance = principal * math.e ** (rate * time)
balance = principal * math.exp(rate * time)
```

- More: `sqrt`, `factorial`, ...

```
https://docs.python.org/3/library/math.html
```

Rounding

One more numeric function, not in the math library: `round`

- Two arguments: the float to round, and an integer number of digits.

Rounding

One more numeric function, not in the math library: `round`

- Two arguments: the float to round, and an integer number of digits.
- Returns a float rounded to that many digits after the decimal.

```
print(round(math.pi, 2))   →   3.14
```

Rounding

One more numeric function, not in the math library: `round`

- Two arguments: the float to round, and an integer number of digits.
- Returns a float rounded to that many digits after the decimal.

```
print(round(math.pi, 2))  →  3.14
print(round(math.e, 0))   →  3.0
print(round(12, -1))      →  10
```

Rounding

One more numeric function, not in the math library: `round`

- Two arguments: the float to round, and an integer number of digits.
- Returns a float rounded to that many digits after the decimal.

```
print(round(math.pi, 2))  →  3.14
print(round(math.e, 0))   →  3.0
print(round(12, -1))      →  10
```

- Note that 0.5 doesn't always round up!
  - Rounds it to the even number—more fair when taking averages.

Rounding

One more numeric function, not in the math library: `round`

- Two arguments: the float to round, and an integer number of digits.
- Returns a float rounded to that many digits after the decimal.

```
print(round(math.pi, 2))   →   3.14
print(round(math.e, 0))   →   3.0
print(round(12, -1))   →   10
```

- Note that 0.5 doesn't always round up!
  - Rounds it to the even number—more fair when taking averages.
- Doesn't change the argument!
  - If you want to do that, use an assignment:

```
area = round(area, 2)
```

Rounding

One more numeric function, not in the math library: `round`

- Two arguments: the float to round, and an integer number of digits.
- Returns a float rounded to that many digits after the decimal.

```
print(round(math.pi, 2))   →   3.14
print(round(math.e, 0))   →   3.0
print(round(12, -1))   →   10
```

- Note that 0.5 doesn't always round up!
  - Rounds it to the even number—more fair when taking averages.
- Doesn't change the argument!
  - If you want to do that, use an assignment:

```
area = round(area, 2)
```

  - But don't do that if you're not done calculating!
    - ★ More precise to save rounding until the very end.
    - ★ Keep the original and round only in the output.

Rounding

One more numeric function, not in the math library: `round`

- Two arguments: the float to round, and an integer number of digits.
- Returns a float rounded to that many digits after the decimal.

```
print(round(math.pi, 2))   →   3.14
print(round(math.e, 0))   →   3.0
print(round(12, -1))   →   10
```

- Note that 0.5 doesn't always round up!
  - Rounds it to the even number—more fair when taking averages.
- Doesn't change the argument!
  - If you want to do that, use an assignment:

```
area = round(area, 2)
```

  - But don't do that if you're not done calculating!
    - ★ More precise to save rounding until the very end.
    - ★ Keep the original and round only in the output.

A complete program

Let's go through the whole process of making a (simple) program, from start to finish.

A complete program

Let's go through the whole process of making a (simple) program, from start to finish. The steps are:

- Specification (the "assignment": usually given to you)
- Test plan.
- Design (pseudocode, algorithm).
- Writing code.
- Testing.

A complete program

Let's go through the whole process of making a (simple) program, from start to finish. The steps are:

- Specification (the "assignment": usually given to you)
- Test plan.
- Design (pseudocode, algorithm).
- Writing code.
- Testing.

Specification

We are given the specification:

> Write a program that asks the user for a temperature in Fahrenheit and converts it to Celsius. The input does not have to be a whole number of degrees. The program should print:
>
> x degrees F is y C.
>
> Use the formula:
>
> $c = \frac{5}{9}(f - 32)$
>
> Round the output to tenths of a degree.

Specification

We are given the specification:

> Write a program that asks the user for a temperature in Fahrenheit and converts it to Celsius. The input does not have to be a whole number of degrees. The program should print:
>
> x degrees F is y C.
>
> Use the formula:
>
> $c = \frac{5}{9}(f - 32)$
>
> Round the output to tenths of a degree.

Test plan

- What kind of inputs to test?

Test plan

- What kind of inputs to test?
- Normal inputs: both integers and floats.

Test plan

- What kind of inputs to test?
- Normal inputs: both integers and floats.
- Are there any boundary cases?

Test plan

- What kind of inputs to test?
- Normal inputs: both integers and floats.
- Are there any boundary cases?
  - Not really.
  - Still, we might test 0, negative numbers, ...

Test plan

- What kind of inputs to test?
- Normal inputs: both integers and floats.
- Are there any boundary cases?
  - Not really.
  - Still, we might test 0, negative numbers, ...
- Other special cases?

Test plan

- What kind of inputs to test?
- Normal inputs: both integers and floats.
- Are there any boundary cases?
  - Not really.
  - Still, we might test 0, negative numbers, ...
- Other special cases?
  - If the input has more than one digit after the decimal point.

Test plan

- What kind of inputs to test?
- Normal inputs: both integers and floats.
- Are there any boundary cases?
  - Not really.
  - Still, we might test 0, negative numbers, ...
- Other special cases?
  - If the input has more than one digit after the decimal point.
- Any error cases?

Test plan

- What kind of inputs to test?
- Normal inputs: both integers and floats.
- Are there any boundary cases?
  - Not really.
  - Still, we might test 0, negative numbers, ...
- Other special cases?
  - If the input has more than one digit after the decimal point.
- Any error cases?
  - Non-numeric input.

Test plan

- What kind of inputs to test?
- Normal inputs: both integers and floats.
- Are there any boundary cases?
  - Not really.
  - Still, we might test 0, negative numbers, ...
- Other special cases?
  - If the input has more than one digit after the decimal point.
- Any error cases?
  - Non-numeric input.

Test plan

Description	Input	Expected output
Normal, integer	32	32.0 F is 0.0 C.
Normal, float	98.6	98.6 F is 37.0 C.
Normal, float answer	57.5	57.5 F is 14.2 C.
Normal, zero	0.0	0.0 F is -17.8 C.
Normal, negative	-40	-40.0 F is -40.0 C.
Special, many digits	0.33333	0.3 F is -17.6 C.
Error, non-numeric	`zero`	*Terminates with error message.*

Design

For the design, we start with the purpose, inputs (preconditions), and outputs (postconditions).

Design

For the design, we start with the purpose, inputs (preconditions), and outputs (postconditions).

- Purpose: Convert a temperature from Fahrenheit to Celsius.

Design

For the design, we start with the purpose, inputs (preconditions), and outputs (postconditions).

- Purpose: Convert a temperature from Fahrenheit to Celsius.
- Preconditions: User enters a temperature in Fahrenheit.

Design

For the design, we start with the purpose, inputs (preconditions), and outputs (postconditions).

- Purpose: Convert a temperature from Fahrenheit to Celsius.
- Preconditions: User enters a temperature in Fahrenheit.
- Postconditions: Program prints the message "x F is y C.", rounded to one digit after the decimal point.

Design

For the design, we start with the purpose, inputs (preconditions), and outputs (postconditions).

- Purpose: Convert a temperature from Fahrenheit to Celsius.
- Preconditions: User enters a temperature in Fahrenheit.
- Postconditions: Program prints the message "x F is y C.", rounded to one digit after the decimal point.

Pseudocode

So how will we accomplish this?

Pseudocode

So how will we accomplish this?

1. Get the Fahrenheit temperature from the user.

Pseudocode

So how will we accomplish this?

1. Get the Fahrenheit temperature from the user.
2. Convert to Celsius using the formula $C = \frac{5}{9}(F - 32)$.

Pseudocode

So how will we accomplish this?

1. Get the Fahrenheit temperature from the user.
2. Convert to Celsius using the formula $C = \frac{5}{9}(F - 32)$.
3. Round the Fahrenheit temperature to one decimal.

Pseudocode

So how will we accomplish this?

1. Get the Fahrenheit temperature from the user.
2. Convert to Celsius using the formula $C = \frac{5}{9}(F - 32)$.
3. Round the Fahrenheit temperature to one decimal.
4. Round the Celsius temperature to one decimal.

Pseudocode

So how will we accomplish this?

1. Get the Fahrenheit temperature from the user.
2. Convert to Celsius using the formula $C = \frac{5}{9}(F - 32)$.
3. Round the Fahrenheit temperature to one decimal.
4. Round the Celsius temperature to one decimal.
5. Output the answer message.

Pseudocode

So how will we accomplish this?

1. Get the Fahrenheit temperature from the user.
2. Convert to Celsius using the formula $C = \frac{5}{9}(F - 32)$.
3. Round the Fahrenheit temperature to one decimal.
4. Round the Celsius temperature to one decimal.
5. Output the answer message.

Note: none of the above steps were Python code!

Pseudocode

So how will we accomplish this?

1. Get the Fahrenheit temperature from the user.
2. Convert to Celsius using the formula $C = \frac{5}{9}(F - 32)$.
3. Round the Fahrenheit temperature to one decimal.
4. Round the Celsius temperature to one decimal.
5. Output the answer message.

Note: none of the above steps were Python code!

Pseudocode in your design should be written so that it could be implemented in any programming language, not just Python.

Pseudocode to comments.

Make each step into a comment.

```
# Purpose: Convert a temperature from Fahrenheit to Celsius.
# Preconditions: User enters a temperature in Fahrenheit.
# Postconditions: Program prints the message "x F is y C.",
#     rounded to one digit after the decimal point.
# 1. Get the Fahrenheit temperature from the user.
# 2. Convert to Celsius using the formula C = 5/9 (F-32)
# 3. Round the Fahrenheit temperature to one decimal.
# 4. Round the Celsius temperature to one decimal.
# 5. Output the answer message.
```

Pseudocode to comments.

Make each step into a comment.

```
# Purpose: Convert a temperature from Fahrenheit to Celsius.
# Preconditions: User enters a temperature in Fahrenheit.
# Postconditions: Program prints the message "x F is y C.",
#     rounded to one digit after the decimal point.
# 1. Get the Fahrenheit temperature from the user.
# 2. Convert to Celsius using the formula C = 5/9 (F-32)
# 3. Round the Fahrenheit temperature to one decimal.
# 4. Round the Celsius temperature to one decimal.
# 5. Output the answer message.
```

Writing the code

Put the steps inside a `def main():`, and call it at the end.

```
# Purpose: Convert a temperature from Fahrenheit to Celsius.
# Preconditions: User enters a temperature in Fahrenheit.
# Postconditions: Program prints the message "x F is y C.",
#    rounded to one digit after the decimal point.
def main():
  # 1. Get the Fahrenheit temperature from the user.
  # 2. Convert to Celsius using the formula C = 5/9 (F-32)
  # 3. Round the Fahrenheit temperature to one decimal.
  # 4. Round the Celsius temperature to one decimal.
  # 5. Output the answer message.
main()
```

Writing the code

Put the steps inside a `def main():`, and call it at the end.

```
# Purpose: Convert a temperature from Fahrenheit to Celsius.
# Preconditions: User enters a temperature in Fahrenheit.
# Postconditions: Program prints the message "x F is y C.",
#    rounded to one digit after the decimal point.
def main():
  # 1. Get the Fahrenheit temperature from the user.
  # 2. Convert to Celsius using the formula C = 5/9 (F-32)
  # 3. Round the Fahrenheit temperature to one decimal.
  # 4. Round the Celsius temperature to one decimal.
  # 5. Output the answer message.
main()
```

Writing the code

And write code for each line of the design.

```
def main():
  # 1. Get the Fahrenheit temperature from the user.
```

Writing the code

And write code for each line of the design.

```
def main():
  # 1. Get the Fahrenheit temperature from the user.
  fahr = float(input("Enter a temp in Fahrenheit: "))
```

Writing the code

And write code for each line of the design.

```
def main():
  # 1. Get the Fahrenheit temperature from the user.
  fahr = float(input("Enter a temp in Fahrenheit: "))
  # 2. Convert to Celsius using the formula C = 5/9 (F-32)
```

Writing the code

And write code for each line of the design.

```
def main():
  # 1. Get the Fahrenheit temperature from the user.
  fahr = float(input("Enter a temp in Fahrenheit: "))
  # 2. Convert to Celsius using the formula C = 5/9 (F-32)
  celsius = (5/9) * (fahr - 32)
```

Writing the code

And write code for each line of the design.

```
def main():
  # 1. Get the Fahrenheit temperature from the user.
  fahr = float(input("Enter a temp in Fahrenheit: "))
  # 2. Convert to Celsius using the formula C = 5/9 (F-32)
  celsius = (5/9) * (fahr - 32)
  # 3. Round the Fahrenheit temperature to one decimal.
```

Writing the code

And write code for each line of the design.

```
def main():
  # 1. Get the Fahrenheit temperature from the user.
  fahr = float(input("Enter a temp in Fahrenheit: "))
  # 2. Convert to Celsius using the formula C = 5/9 (F-32)
  celsius = (5/9) * (fahr - 32)
  # 3. Round the Fahrenheit temperature to one decimal.
  fahr_round = round(fahr, 1)
```

Writing the code

And write code for each line of the design.

```
def main():
  # 1. Get the Fahrenheit temperature from the user.
  fahr = float(input("Enter a temp in Fahrenheit: "))
  # 2. Convert to Celsius using the formula C = 5/9 (F-32)
  celsius = (5/9) * (fahr - 32)
  # 3. Round the Fahrenheit temperature to one decimal.
  fahr_round = round(fahr, 1)
  # 4. Round the Celsius temperature to one decimal.
```

Writing the code

And write code for each line of the design.

```
def main():
  # 1. Get the Fahrenheit temperature from the user.
  fahr = float(input("Enter a temp in Fahrenheit: "))
  # 2. Convert to Celsius using the formula C = 5/9 (F-32)
  celsius = (5/9) * (fahr - 32)
  # 3. Round the Fahrenheit temperature to one decimal.
  fahr_round = round(fahr, 1)
  # 4. Round the Celsius temperature to one decimal.
  cels_round = round(celsius, 1)
```

Writing the code

And write code for each line of the design.

```
def main():
  # 1. Get the Fahrenheit temperature from the user.
  fahr = float(input("Enter a temp in Fahrenheit: "))
  # 2. Convert to Celsius using the formula C = 5/9 (F-32)
  celsius = (5/9) * (fahr - 32)
  # 3. Round the Fahrenheit temperature to one decimal.
  fahr_round = round(fahr, 1)
  # 4. Round the Celsius temperature to one decimal.
  cels_round = round(celsius, 1)
  # 5. Output the answer message.
```

Writing the code

And write code for each line of the design.

```
def main():
  # 1. Get the Fahrenheit temperature from the user.
  fahr = float(input("Enter a temp in Fahrenheit: "))
  # 2. Convert to Celsius using the formula C = 5/9 (F-32)
  celsius = (5/9) * (fahr - 32)
  # 3. Round the Fahrenheit temperature to one decimal.
  fahr_round = round(fahr, 1)
  # 4. Round the Celsius temperature to one decimal.
  cels_round = round(celsius, 1)
  # 5. Output the answer message.
  print(fahr_round, "F is", cels_round, "C.")
```

Writing the code

And write code for each line of the design.

```
def main():
  # 1. Get the Fahrenheit temperature from the user.
  fahr = float(input("Enter a temp in Fahrenheit: "))
  # 2. Convert to Celsius using the formula C = 5/9 (F-32)
  celsius = (5/9) * (fahr - 32)
  # 3. Round the Fahrenheit temperature to one decimal.
  fahr_round = round(fahr, 1)
  # 4. Round the Celsius temperature to one decimal.
  cels_round = round(celsius, 1)
  # 5. Output the answer message.
  print(fahr_round, "F is", cels_round, "C.")

main()
```

Testing

- Now run the program once for each test case.

Testing

- Now run the program once for each test case.
- Give the input and verify that the output matches.

Testing

- Now run the program once for each test case.
- Give the input and verify that the output matches.
- If not, there is a bug:
  - Maybe in your program...
  - Maybe in your test case!

Testing

- Now run the program once for each test case.
- Give the input and verify that the output matches.
- If not, there is a bug:
  - Maybe in your program...
  - Maybe in your test case!
- After you fix a bug, repeat all the tests.

Testing

- Now run the program once for each test case.
- Give the input and verify that the output matches.
- If not, there is a bug:
  - Maybe in your program...
  - Maybe in your test case!
- After you fix a bug, repeat all the tests.
  - Regression testing.

Testing

- Now run the program once for each test case.
- Give the input and verify that the output matches.
- If not, there is a bug:
  - Maybe in your program...
  - Maybe in your test case!
- After you fix a bug, repeat all the tests.
  - Regression testing.

The end

Next time:

- The `graphics` library.
  - Not in the textbook!
- Conditions: deciding which code to run.