CS 115 Libraries, F to C

Taken from notes by Dr. Neil Moore

The math library

We’ve seen 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 trigonometry, logarithms, and more.
  - Constants like π and *e*.

Using libraries in Python

- To use a library, you must first **import** it.

  ```
  import math
  ```

  - Put this at the very top of the program
  - After header comments, before "`def main():`"
- Then your program can use the things in the library
  - Their names are *library.name*
  - So `math.log` (function) and `math.pi` (constant)
  - You call functions with parenthesized arguments
    - Just like `input` and `print`
  - Each function has its own rules about what its arguments are, what they mean, how many there are, etc.

Using libraries in Python

- If the function returns a value, you use it as part of an expression

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

- If it does not return a value, use it as an entire statement by itself:

```
random.seed()
```

- Only import the libraries you need!
  - for documentation, for efficiency, for style

A variation on import

- You can instead import particular functions or constants specifically by writing import this way:

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

  - List the names that you are importing, separated by commas
  - Then you can use them without the “math.”

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

  - Saves typing if you use a function many times

One last variation on import

- One last way to write import:

  ```
  from math import *
  ```

  - It imports everything in the library
  - And you don’t have to use “math.”

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

  - Sounds great, right? There can be a catch…
  - What if next version of Python adds a new function which is the same name as one of your variables or functions?
  - Your code could break! And have to be rewritten!
  - Professional programmers avoid “from lib import *, because of this catch. In class we’ll use it occasionally

What’s in the math library

- Trigonometry: sin, cos, tan, cosh,…
  - `angle = math.atan(a/b)`
  - `circumference = math.pi * diameter`
- Natural logarithm and other bases:
  - `doubling_time = math.log(2) / rate`
  - `pH = -log(activity, 10)`
- e and e^x
  - `balance = principal * math.e ** (rate * time)`
  - `balance = principle * math.exp(rate* time)`
- More: `sqrt, factorial, fib,` …
- `https://docs.python.org/3/library/math.html`

Common misunderstanding

- For some reason, once people know about the math library, they feel that they MUST import it for any kind of arithmetic, using +, -, *, /, //, **, %, etc.
- This is NOT the case. All these operators were available before you even knew about import, they are still available as being builtin to Python.
- You need to import math ONLY when you are using math library **functions (sqrt, log, …)** and **constants (pi, e)**

Rounding

One more numeric function, builtin – so you do NOT have to import math library to use it

- round has **either** one or two arguments
  - If it has just ONE argument, it will round the argument to the nearest integer
    - round(5.2) → 5
    - round (7.9) → 8
  - If it has TWO arguments, the second one is the number of decimal places desired. The first argument's value will be rounded to that number of decimals
    - round (math.pi, 2) → 3.14
    - round (2.71818, 0) → 3.0
    - round (12, -1) → 10

Rounding

One more numeric function, builtin – so you do NOT have to import math library to use it

- round has **either** one or two arguments
  - If it has just ONE argument, it will round the argument to the nearest integer
    - round(5.2) → 5
    - round (7.9) → 8
    - round (5.235) → 5
    - round (5.725) →6

Rounding

One more numeric function, builtin – so you do NOT have to import math library to use it

- round has **either** one or two arguments
  - If it has TWO arguments, the second one is the number of decimal places desired. The first argument's value will be rounded to that number of decimals
    - round (math.pi, 2) → 3.14
    - round (2.71818, 0) → 3.0
    - round (12, -1) → 10
- Note that a value ending in 5 does not always round up! It rounds towards the even number – doc.python.org says that it is because of the problems with representing floating point numbers
    - So that round(5.3545, 3) → 5.354 (because 4 is even)
    - And round(5.3555, 3) → 5.356 (because 6 is even)
    - And round(4.5) → 4, and round(5.5) → 6

The round function

- **round(*number*[, *ndigits*])** Return the floating point value *number* rounded to *ndigits* digits after the decimal point. If *ndigits* is omitted, it returns the nearest integer to its input.
- For the built-in types supporting round(), values are rounded to the closest multiple of 10 to the power minus *ndigits*; if two multiples are equally close, rounding is done toward the even choice (so, for example, both round(0.5) and round(-0.5) are 0, and round(1.5) is 2). The return value is an integer if called with one argument, otherwise of the same type as *number*.
- The behavior of round() for floats can be surprising: for example, round(2.675, 2) gives 2.67 instead of the expected 2.68. This is not a bug: it's a result of the fact that most decimal fractions can't be represented exactly as a float. See Floating Point Arithmetic: Issues and Limitations for more information.
- From: https://docs.python.org/3/library/functions.html?highlight=round%20function#round

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 answer to tenths of a degree.

Test plan

- What kind of inputs to test?
- Normal inputs: both integers and floats.
- Are there any boundary cases?
  - Not really for the **formula**
    - Some people would argue for absolute zero (-459.67 degrees Fahrenheit or 273.15 degrees Celsius) because of **physics**
  - Still we might test 0, should test negative numbers
- Other special cases?
  - If the input had more than one digit after decimal, to check for rounding correctly
- 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, zero	0.0	0.0 F is -17.8 C
Normal, negative	-40	-40.0 F is -40.0 C
Normal, absolute zero	-459.67	-459.67 F is -273.2 F
Special, to check rounding	0.33333	0.3 F is -17.6 C
Error, non-numeric input	Zero	Terminates with error message about wrong type

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?

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 was 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.
```

Writing the code

Put the steps inside a `def main():` and call the main function 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 form. 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
  fahr = float(input(“Enter  a temp in Fahrenheit: “))
  # 2. Convert to Celsius using the form. 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.
- Give the input and verify that the output matches the expected output.
- 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!