Welcome!

We'll get started shortly. Please take the Zoom poll in the meanwhile!

CS 49 Week 4

Surajit A. Bose

Agenda

- Console input and output
- Variables and expressions
- Worked example: Fahrenheit to Celsius
- Constants
- External modules and libraries
- Worked example: Pythagorean Theorem
- Section problem: Mars Weight
- Rounding decimal numbers

How to get hold of me / get help from other resources

- Surajit's office hours
  - Fridays 12 noon–1p, directly after section
  - By appointment on Zoom
- Pronto DM for a quick response (usually within a couple hours)
- Email bosesurajit@fhda.edu or Canvas inbox, 24 hr turnaround
- The github repo has section materials, starter code, and solutions

---

- Pronto DM or Canvas inbox for Lane
- Lane's office hours
- Online or in-person tutoring at the STEM center (Room 4213)

Console Input and Output

The console, aka the terminal

Console output via `print()`

- The console is where Python programs display their output
- The command used for console output is print()
- Sample program in Python that prints **`hello, world`** to the screen:

```
def main()
    print("hello, world")

if __name__ == "__main__":
    main()
```

- The output **`hello, world`** will be printed to screen without the quotation marks

Getting user input at the console

- The program can get information from the user via the **`input()`** command
- The parentheses contain the prompt that is printed onscreen
- The program prints the **prompt** and waits for the user to enter something
- Sample input() command asking for the user name:

```
user_name = input("What is your name? ")
```

- The prompt is inside quotation marks
- For good formatting, put a space at the end of the prompt (before the close quote)
- The user input needs to be stored in a **variable** for the program's use
- Here, **`user_name`** is the variable that stores what the user enters at the prompt

Variables and Expressions

Variables

- A variable is a **named** place in memory that holds a **value** of a particular **type**
- A variable is a **location in memory**
- The variable **is** or **has** a **name**. The name is an identifier or tag that specifies the memory location
- The variable **has** or **holds** a **value**, such as 8.04, -16, or "steve@apple.com"
- The value is of a specific **type**. The basic or primitive types in Python are:
  - a `float` (a number with decimal places, could be positive or negative)
  - an `int` (an integer, could be positive or negative)
  - a `str` (a string, a sequence of characters enclosed in single or double quotes)
  - a `bool` (a boolean, for two specific values `True` and `False` used when evaluating conditions)
- Variables are **assigned** using a single equal sign, e.g., `num_planets = 8`

Arithmetic expressions (Slide 1 of 3)

- An expression is a statement that can be evaluated
- With Karel, we've seen boolean expressions such as **`front_is_clear()`** or **`beepers_present()`** that evaluate to **`True`** or **`False`**
- There are also arithmetic expressions: Given **`x = 9`** and **`y = 2`**,

```
z = x + y       # z = 11
z = x - y       # z = 7
z = x * y       # z = 18
z = x / y       # z = 4.5
```

```
z = x // y      # z = 4
z = x % y       # z = 1
z = x ** y      # z = 81
z = -y          # z = -2
```

Arithmetic expressions (Slide 2 of 3)

- The symbols **+**, *****, **/**, etc. are the **operators**
- The terms operated on (**x** and **y** in the previous slide) are the **operands**
- Typically, the result or value of an arithmetic expression is stored in a variable (**z** in the previous slide)
- Keep in mind the difference between these three operators:
  - **/** will always result in a float
  - **//** will always result in an integer, any remainder being discarded
  - **%** is the modulus operator for the remainder of integer division
- Given **x = 8** and **y = 2**, what is the value and type of these expressions?

```
x ** y  # ?        x / y  # ?        x < y  # ?
```

Arithmetic expressions (Slide 2 of 3)

- The symbols `+`, `*`, `/`, etc. are the **operators**
- The terms operated on (`x` and `y` in the previous slide) are the **operands**
- Typically, the result or value of an arithmetic expression is stored in a variable (`z` in the previous slide)
- Keep in mind the difference between these three operators:
  - `/` will always result in a float
  - `//` will always result in an integer, any remainder being discarded
  - `%` is the modulus operator for the remainder of integer division
- Given `x = 8` and `y = 2`, what is the value and type of these expressions?

```
x ** y  # 64        x / y  # 4.0        x < y  # False
```

More about types

- All console input and output is done with type `str`
- When the user inputs a number, we need to convert it from `str` to `float` or `int` as appropriate
- This is done by **casting** the input as the appropriate type
- Conversely, when the program prints numbers to screen, they need to be converted to `str`
- This is done by using **f-strings**, short for **format strings**, a mechanism in Python that automatically casts variables to `str` for output
- Let's try this out in a program that asks the user for a temperature in Fahrenheit and prints out the same temperature in Celsius

Worked Example: Converting Temperatures

Worked example: Fahrenheit to Celsius

Write a program which prompts the user for a temperature in Fahrenheit (this can be a number with decimal places!) and outputs the temperature converted to Celsius.
The Celsius scale is widely used to measure temperature, but places like the US still use Fahrenheit. Fahrenheit is another unit for temperature, but the scale is different from Celsius -- for example, 0 degrees Celsius is 32 degrees Fahrenheit!

The equation you should use for converting from Fahrenheit to Celsius is the following:
degrees_celsius = (degrees_fahrenheit - 32) * 5.0/9.0

(Note: The .0 after the 5 and 9 matters in the line above!!!)

Here's a sample run of the program (user input is in bold italics):

> Enter temperature in Fahrenheit: ***76***
> Temperature: 76.0F = 24.444444444444443C

Steps to the solution

- Get input from the user and store it in a variable
  - What would be a good variable name?
  - What type will the variable be?
- Cast the variable to the appropriate type
  - As what type should the value be cast?
- Apply the conversion formula and store the result in a variable
  - What would be a good variable name?
- Use an f-string to print the result to the screen

Constants

Constants

- In Python, a constant is a variable whose value does not change during the execution of the program
- By convention, constants are named in **`UPPER_SNAKE_CASE`**
- Why use constants?
  - To avoid "magic numbers"
  - To allow easy updates
  - To follow the principle of programming for the general case
- Unlike most other programming languages, Python does not enforce constants; they are a convention

Modules and Libraries

The `random` module

- A module is a python file containing pre-existing code that provides useful functionality
- The `random` module which allows us to generate pseudo-random numbers
- `random.randint(x, y)` will generate an `int` between `x` and `y` inclusive
- `random.random()` will generate a `float` between 0 and 1, not including 1
- `random.uniform(x, y)` will generate a `float` between `x` and `y` inclusive
- For debugging, `random.seed(x)` will set the seed for the random generator to a specified value `x` so that the sequence of random numbers generated is the same each time the program is run

The `math` and `ai` libraries

- Loosely speaking, a library is a collection of many modules
- The `ai` library provides an interface to ChatGPT
- The `math` library provides functionality such as calculating square roots
- To use such external libraries in our own programs, we need to use an `import` statement such as `from ai import call_gpt`, `import random`, or `import math`
- We've seen such a statement before: `from karel.stanfordkarel import *`
- When using a statement like `import math` or `import random`, we have to use the name of the library or module in the command. For example, the command to calculate the square root of 25 is `math.sqrt(25)`

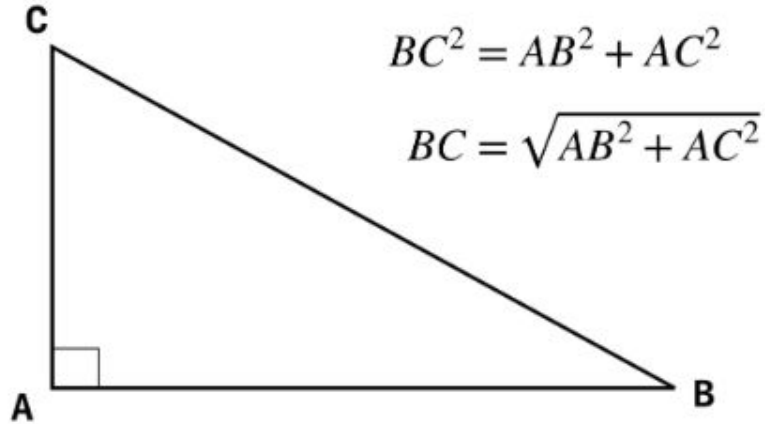
The structure of a Python program

- Python programs have a typical order:
  - Comment with filename, program overview, and programmer name
  - **`import`** statements
  - constants
  - **`main()`** function
  - helper functions
  - guard clause and invocation of **`main()`**
- A template for your use is on the section github repo

Section Problem: Pythagorean Theorem

Pythagorean theorem

Write a program that asks the user for the lengths of the two perpendicular sides of a right triangle and outputs the length of the third side (the hypotenuse) using the Pythagorean theorem! The Pythagorean theorem, named after the ancient Greek thinker, Pythagoras, is a fundamental relation in geometry. It states that in a right triangle, the square of the hypotenuse is equal to the sum of the square of the other two sides.



$$BC^2 = AB^2 + AC^2$$

$$BC = \sqrt{AB^2 + AC^2}$$

Steps to solution

- Get len_AB from user and convert to appropriate type
- Get len_AC from user and convert to appropriate type
- Compute sum of squares
- Compute len_BC
- Display len_BC

Here's a sample run of the program (user input is in bold italics):

> Enter the length of AB: ***3***
> Enter the length of AC: ***4***
> The length of BC (the hypotenuse) is: 5.0

Section Problem: Mars Weight

Mars Weight

Due to the weaker gravity on Mars, an Earthling's weight on Mars is 37.8% of their weight on Earth. Write a Python program that prompts an Earthling to enter their weight on Earth and prints their calculated weight on Mars. The output should be rounded to two decimal places when necessary. Example:

```
Enter a weight on Earth: 120
The equivalent weight on Mars: 45.36
```

- What constant should we use?
- As what type should the input from the user be cast?
- How to round the output to two decimal places?

Built-in functions: `round()`

- In addition to the functionality provided by modules and libraries, Python also has its own suite of built-in functions
- One such function is **`round()`**, used to round a **`float`** to the nearest **`int`** or to a specified number of decimal spaces
- Example:

```
round(3.1415)      # result: 3
round(3.1415, 3)   # result: 3.142
```

That's all, folks!

Next up: Control Flow in Python

Bonus Slides

More about variables and operators

More about variable names

Variable names (Slide 1 of 3)

- A variable is a **named** place in memory that holds a **value** of a certain **type**
- Names are case sensitive: `cumulative_GPA` is different from `cumulative_gpa`
- Restrictions on names (enforced by Python):
  - Must begin with a letter or an underscore
  - Must not be a reserved word like `for` or `def`
- Warning about names (not enforced by the language):
  - Should not replicate the name of a built-in function like `print` or `round`
- Conventions about names (style guidelines):
  - Should be short but descriptive
  - Should be in `snake_case` if longer than one word long

Variable names (Slide 2 of 3)

- Which of the following variable names meet the specified standards?
  - Must begin with a letter or underscore[1]
  - Must not be a reserved word such as **`for`** or **`def`**[1]
  - Should not replicate names for built-in functions like **`print()`**[2]
  - Should be in **`snake_case`** if more than one word long[3]
  - Should be short but descriptive[3]
- [1]illegal in Python to violate this; [2]legal but extremely inadvisable; [3]recommended convention

`result 101_dalmatians num_students pass input numStudents longitude y total main 1atitude`

Variable names (Slide 3 of 3)

- Which of the following variable names meet the specified standards?
  - Must begin with a letter or underscore[1]
  - Must not be a reserved word such as **`for`** or **`def`**[1]
  - Should not replicate names for built-in functions like **`print()`**[2]
  - Should be in **`snake_case`** if more than one word long[3]
  - Should be short but descriptive[3]
- **`red`**: illegal in Python to violate this; **`purple`**: legal but extremely inadvisable; **`brown`**: against recommended convention; **`green`**: fine

```
result   101_dalmatians   num_students   pass   input
numStudents   longitude   y   total   main   1atitude
```

More about variable values

Variable value (Slide 1 of 3)

- A variable is a **named** place in memory that holds a **value** of a particular **type**
- Value:
  - Assigned to a variable with the equals sign, e.g. `answer = 42`
  - This is also called "binding": the value `42` is bound to the variable `answer`
  - The assigned value can be an expression, e.g.
    ```
    gpa = qual_points / num_credits
    ```
  - The right hand side of the equals sign is **evaluated** , then the result is placed into the variable on the left hand side
  - This means we can have commands like
    ```
    balance = balance + interest
    ```

Variable value (Slide 2 of 3)

- As we saw with **`input()`**, a value can be **cast** to a different type
- Given **`int_val = 3`**, what would the results be of the following cast operations?

```
float_val = float(int_val)
str_val = str(int_val)
str_float_val = str(float_val)
int_val_2 = int(str_float_val)

e = 2.7183
int_e = int(e)
```

Variable value (Slide 3 of 3)

- As we saw with **`input()`**, a value can be **cast** to a different type
- Given **`int_val = 3`**, what would the results be of the following cast operations?

```
float_val = float(int_val)        # result: 3.0
str_val = str(int_val)            # result: '3'
str_float_val = str(float_val)    # result: '3.0'
int_val_2 = int(str_float_val)    # result: error

e = 2.7183
int_e = int(e)                    # result: 2
```

More about variable types

Variable type (Slide 1 of 3)

- With arithmetic expressions, the type of the result depends on both the operator and the operands
  - `/` always results in a `float`, `//` always in an `int`
  - All the others will result in an `int` if both operands are `int`s, or a `float` if either of the operands is a `float`
- What are the types of the following values?

```
35      beepers_present()      8.13        53 + 72
12 * 6      "7 + 11"        12 / 6.0        12 / 6
'front_is_clear()'          wage_rate * hours_worked
```

Variable type (Slide 2 of 3)

- With arithmetic expressions, the type of the result depends on both the operator and the operands
  - `/` always results in a **`float`**, `//` always in an **`int`**
  - All the others will result in an int if both operands are **`int`**s, or a **`float`** if either of the operands is a **`float`**
- Types: **`red`**, **`str`**; **`green`**, **`int`**; **`purple`**, **`float`**; **`brown`**, **`bool`**; **`pink`**, can't say

```
35          beepers_present()          8.13          53 + 72
12 * 6          "7 + 11"          12 / 6.0          12 / 6
'front_is_clear()'          wage_rate * hours_worked
```

Variable type (Slide 3 of 3)

- Watch out for floating point values! They are not stored precisely:

```
x = 1.9
y = 1
z = x - y
print(z)                    # 0.89999999999999
```

- Precision of **float**s is not reliable beyond the precision determinable by the inputs. Here, the value of **z** is not reliable past one decimal place.
- Can use **round(a, b)** where **a** is the value to round, **b** the number of decimal places:

```
print(round(z, 1))      # 0.9
```

More about operators

Operator Precedence

- Operators have the following precedence:

( )	parentheses
******	exponentiation
-	unary negation
***, /, //, %**	multiplication, division, integer division, modulus
+, -	addition, subtraction

- Operators with the same precedence (e.g., multiplication, division) are evaluated from left to right

Compound Operators (Slide 1 of 2)

- Compound operators: **+=**, **-=**, ***=**, etc. combine the arithmetic and assignment operators in a single command
- Given initial values **x = 3** and **y = 2**, what would the following expressions evaluate to?

```
x *= y      # x = x * y
x += 4      # x = x + 4
x /= y      # x = x / y
x %= y      # x = x % y
```

Compound Operators (Slide 2 of 2)

- Compound operators: **+=**, **-=**, ***=**, etc. combine the arithmetic and assignment operators in a single command
- Given initial values **x = 3** and **y = 2**, what would the following expressions evaluate to?

```
x *= y      # x = x * y     x = 6
x += 4      # x = x + 4     x = 10
x /= y      # x = x / y     x = 5.0
x %= y      # x = x % y     x = 2.0
```

- Notice that the type of the result depends on both the operators and the operand