Week 1 - Wednesday

COMP 1800

Last time

- What did we talk about last time?
- Syllabus
- Problem solving

Questions?

About Python

- Developed by Guido van Rossum, beginning in 1989 in the Netherlands
  - Named after Monty Python's Flying Circus
- First version was released in 1991 (v0.90)
  - Object-oriented
- Version 1.0: 1994
  - Functional programming
- Version 2.0: 2000
  - List comprehensions and garbage collection
- Version 3.0: 2008
  - Fixed fundamental language design flaws

Guido van Rossum, 2006
"Benevolent dictator for life" until 2018

What is an object?

- In Python, everything is an object
- But what is an object?!
- Four properties:
  - Identity: Similar to a name
  - State: Values inside
  - Type: Classification of the object
  - Behavior: Stuff an object can do with its state, called methods

Programs

- According to the object-oriented perspective, a **program** is a specification of objects and how they interact
- Another way of looking at a program is a sequence of characters (Unicode, ASCII, other) that follow certain rules of structure

Linguistics

- English is composed of sentences made up of words that follow a certain grammar
  - *<subject> <predicate>*
- Python is composed of programs made up of **tokens** that follow a certain grammar
- There are five kinds of tokens in most programming languages

Tokens

1. Literals
   - Values from some built-in type
2. Identifiers
   - Names used for things
3. Keywords
   - Identifiers with special meaning
4. Operators
   - Computational actions
5. Separators
   - Punctuation

- White space also plays a role, as do comments

Examples with IDLE

IDLE

- I will use IDLE for all examples in class
- There are other Python IDEs, some with more power, but IDLE comes with Python
- You can use IDLE interactively
  - Every line you type in is executed
  - Called a read-eval-print loop (or REPL)
- You can also have IDLE run a file that was written earlier
  - This is how you will turn in most of your assignments
- Both ways of running IDLE are useful

Literals

- Each type has a number of **literals** associated with it
- A **literal** is a concrete value within a given type
- **Literals** for the integer type are numbers exactly like what you would expect:
  - `115`
  - `-9837461`
  - `2`

Let's start by using IDLE as a REPL

- We can type literals and operations and see what the results are

```
>>> 5
    5
>>> 4 - 10
    -6
```

Variables

- It is also possible to create **variables** for each data type
- Think of a variable as a "box" that you can put values into
- The name of a variable is an **identifier**
- The type of a variable is whatever you've most recently put into it
- The following creates a variable called **`i`** that currently holds an integer
- Then, we multiply **`i`** by 3

```
>>> i = 42
>>> 3 * i
    126
```

Changing the value of a variable

- Python variables are not like variables in math which have a fixed (but unknown) value
- Instead, a Python variable can be changed by a line of code
- We use the **assignment operator (=)** to change the value of a variable as follows:

```
>>> i = 42
>>> 3 * i
    126
>>> i = 5
>>> 3 * i
    15
```

- The first time, `3 * i` is 126, but the second time, it's 15

Variable names

- Mathematicians often use really short variable names like *x* or *y*
- In Python, the variable names can be longer
- Choose name that are descriptive
- Variables have to start with a letter (or an underscore) and then can have letters, underscores, or numbers

```
>>> rainfall = 2.61
>>> top10 = 21
>>> 5fingers = 19
>>> first number = 42
```

Legal

Illegal

- Spaces aren't allowed in a variable name
- The book uses camel-case
  - To make it more readable, each word in a multi-word variable name is capitalized

```
>>> awesomeVariable = 15
```

Operators

- Python has a number of operators that work with integers and floating-point (decimal) numbers

Operator	Operation	Example	Result
`+`	Add	`5 + 7`	`12`
`-`	Subtract	`5 - 7`	`-2`
`*`	Multiply	`5 * 7`	`35`
`//`	Integer division (round down)	`5 // 7`	`0`
`/`	Regular division	`5 / 7`	`0.7142857142857143`
`%`	Modulo (remainder)	`5 % 7`	`5`
`**`	Exponentiation	`5 ** 7`	`78125`

Order of operations

- You can do complicated expressions
- Just like math, there's an order of operations that determines which operations happen first

Operator	Operation	Evaluation Order
`()`	Parentheses	Left to right
`**`	Exponentiation	Right to left
`* // / %`	Multiplication and division	Left to right
`+ -`	Addition and subtraction	Left to right

```
>>> 3 * (4 + 2) / 8
    2.25
```

Floating-point numbers

- You'll use integers often in programming
  - They're surprisingly versatile!
- Sometimes, however, you need to represent numbers with a fractional part
- Such numbers are called **floating-point numbers** in computer science, because they store a representation of a number with a floating (moving) decimal point
- Integers in Python are exact, but floating-point numbers are approximate
- You can write them normally or in scientific notation where a number after e means multiplied by 10 raised to that number

```
>>> 3.14159
    3.14159
>>> 1.75e7
    17500000.0
```

Converting between integers and floating-point

- In math, there's no difference between 3 and 3.0
- In Python, the difference is there, but it's subtle
- You can convert between the integers and floating-point variables using the following functions

Function	Description	Example	Result
`float(number)`	Convert to floating-point	`float(15)`	`15.0`
`int(value)`	Convert to integer, dropping fractional part	`int(2.7)`	`2`
`round(value)`	Round to the nearest integer	`round(2.7)`	`3`

A Few Python Details

Output

- Basic output is done with **`print()`**
- Put what you want to print inside the parentheses
- You can print:
  - Any text enclosed in single or double quotes:

```
print('43 eggplants')
```

  - Any integer:

```
print(43)
```

  - Any floating-point number:

```
print(23.984)
```

  - Even complex numbers:

```
print(5 + 7j)
```

- If you want multiple things to go on the same line, you can use **`print()`** with more than one argument:

```
print(99, 'red', 'balloons')
```

- By default, they will be printed with a space between each one

Sequencing

- Instead of one print statement, we can have several:

```
print('Hello, world!')
print('Hello, galaxy!')
print('Goodbye, world!')
```

- Each statement is an instruction to the computer
- They are printed in order, one by one

Case Sensitivity

- Python is a case sensitive language
- **`Print`** is not the same as **`print`**
- **`print('Word!')`** prints correctly
- **`Print('Word!')`** causes an error

Whitespace

- Python doesn't care about whitespace **within** a line of code

```
print('Hello, world!')
```

is the same as:

```
print     (    'Hello, world!'    )
```

- However, whitespace at the **beginning** of a line of code **matters**!
- The following will cause an error:

```
    print('Hello, world!')
```

- Indentation is important in Python, so don't indent without reason!

Comments

- Programs can be confusing
- Sometimes you want to leave notes for yourself or anyone else who is reading your code
- The standard way to do this is by using comments
- Although comments appear in the code, they do not affect the final program

Comments

- Single line comments use **#**
- Everything after the **#** is a comment and doesn't affect the execution of the program

```
print('Hi!') # this is a comment
```

- Sometimes, you want to comment out a section of code to see what happens if it doesn't run
- To do that in Python, put triple apostrophe (`'''`) on a line by itself before the code and another after

```
'''
print('Hi!')
print('Bye!)
print('No one will see this!')
'''
```

Turtle

Turtle

- Turtle is a tool that lets us draw simple pictures in Python
- To use Turtle, we first have to import the turtle library

```
>>> import turtle
```

- Then, we have to create a turtle
- I name mine **yertle**, but you can name it any legal variable

```
>>> yertle = turtle.Turtle()
```

- Don't worry too much about this syntax

Methods

- A turtle object has **methods**
- Methods let us tell the turtle to do things or ask it questions
- To call a method, you say the name of the turtle (**`yertle`**, in my case), you put a dot, then you put the name of the method you want, then parentheses, and sometimes information between the parentheses
  - The extra information are called **parameters**
- For example, to make **`yertle`** move forward 100 steps, type:

```
>>> yertle.forward(100)
```

Turtle methods

- The book has a much longer list, but here are a few useful turtle methods

Method	Parameter	Description
`forward`	Distance	Move forward
`backward`	Distance	Move backward
`left`	Angle	Turn counter-clockwise
`right`	Angle	Turn clockwise
`up`	None	Pick up the turtle's tail (to stop drawing)
`down`	None	Put down the turtle's tail (to draw again)
`heading`	None	Return the angle the turtle is pointing
`position`	None	Return the position of the turtle

Turtle practice

- Get the turtle to draw a square whose center is the center of the drawing space

Upcoming

Next time…

- Functions
- `for` loops
- Then, a work day for the first assignment

Reminders

- Review Chapter 1 of the textbook
- Come to class ready to program in IDLE