Lecture 2 — Python as a Calculator

Overview

Most of this is covered in Chapter 2 of *Practical Programming*

- Part 1:
  - Expressions and values
  - Types
  - Precedence
- Part 2:
  - Variables and memory
  - Errors
  - Typing directly in the interpreter vs. running programs; use of *print*
  - Documentation and variable names

Throughout we will pay attention to problems and mistakes, both real and potential.

Aside: Lectures, Note Taking and Exercises

- Lecture notes are outlines.
  - It will help you understand more if you read the notes and work through the examples before class
  - Use class time to develop a more thorough understand and to clarify difficult issues
  - Hand-write details as we cover them
  - If we don't fully cover something that you have a question on, make sure to ask about it, or research it in your text or online
- We will create and run examples in class.
  - You should write down the shorter ones
  - We will post the longer ones on-line, but you should write down as much as you can, especially the results of running the examples. If you don't know what to expect, you can't debug errors.

Python As a Calculator

We will start class by using Python as an interactive calculator, working through a number of examples.

- The area of a circle.
- The number of minutes in a year.
- The volume of a box.
- The volume of the earth in cubic kilometers.
- In doing so, we will look at the basic operations of `+`, `-`, `*`, `/`, and `**` (exponentiation).

Whole Number Calculations

- Most of the calculations in the foregoing examples involve numbers with fractional values.
- Sometimes we want to use whole number of calculations, for example to convert a number of minutes to a number of hours and minutes (less than 60).
- In this case, Python offers us different forms of divisions
    - // is the operator for whole number division
    - % is used to find the remainder
- For this kind of division, be careful of negative numbers. Can you figure out how they work with // and %?
    - This is a case where experimentation with the Python interpreter is crucial. We will encourage you to do this throughout the semester.
    - **Hint:** Consider the 4 cases:
        - 8 and 3
        - 8 and -3
        - -8 and 3
        - -8 and -3

Python Types

- We have seen what look like real numbers and whole numbers. These are two different *types* in Python.
- A type is a set of possible values — also called a "representation" — and a set of operations on those values.
    - Our first examples are the "operators" such as `+`, `-`, `*`, `/` and `**`
- Common Python types we are working with initially include `float`, `int` (short for integer) and `str` (short for string)
- Each value we create will be referred to as an *object*

float

- The `float` type approximates real numbers. But computers can't represent all of them because computers only dedicate a finite amount of memory to each.
- Limited precision: we'll look at 2/3, 5/3 and 8/3
- Any time integers and floats are mixed and any time we apply division, the result is always a float.

int

- The `int` (integer) type is analogous to whole numbers:

```
{ ..., -4, -3, -2, -1, 0, 1, 2, 3, 4, ...}
```

- Python can represent seemingly arbitrarily large integers, which is not true of other languages like C, C++ and Java
  - Using the Python interpreter, we will look at the examples of:

```
>>> 1**1**1
>>> 2**2**2
>>> 3**3**3
```

Precedence

- Consider the formula for converting 45 degrees Fahrenheit to Celsius. What is wrong with the following computation?

```
>>> 45 - 32 * 5 / 9
```

- Largely following the standard rules of algebra, Python applies operations in order of *precedence*. Here is a summary of these rules from highest to lowest:
  1. `( )` - parentheses
  2. `**` - the exponentiation operator, ordered *right-to-left*
  3. `-` - the negation (unary minus) operators, as in `-5**2`
  4. `*, /, //, %` - ordered *left-to-right*
  5. `+, -` - ordered *left-to-right*

- This example also suggests an important question: how do we know we've made a mistake - introduced a *bug* - in our code?

Part 1 Practice Problems

1. The consider the following evaluations (some of them trivial). Which results are `float` objects and which are `int` objects?

```
>>> 9           # 1
>>> 9.          # 2
>>> 9.0         # 3
>>> 9 + 3       # 4
>>> 9 - 3.      # 5
>>> 9 / 4       # 6
>>> 9 // 4      # 7
>>> 9. // 4     # 8
```

2. What is output by the Python interpreter?

```
>>> 2**3**2
>>> (2**3)**2
>>> -2**3 - 2 * 5
```

3. Write a single line of Python code that calculates the radius of a circle with area 15 units and prints the value. The output should just be the number that your code produces. Your code should include the use of an expression involving division and exponentiation (to compute the square root). Use the value 3.14159 for *pi*.

Part 2

Variables and Assignment

- Most calculators have one or several memory keys. Python, and all other programming languages, use "variables" as their memory.
- We'll start with a simple example of the area of a circle, typed in class. You will notice as we go through this that there is no output until we use the *print* functions.
- Here is a more extensive example of computing the volume and surface area of a cylinder:

```
>>> pi = 3.14159
>>> radius = 2
>>> height = 10
>>> base_area = pi * radius ** 2
>>> volume = base_area * height
>>> surface_area = 2 * base_area + 2 * pi * radius * height
>>> print("volume is", volume, ", surface area is", surface_area)
volume is 125.6636 , surface area is 150.79632
```

- A variable is a name that has a value "associated" with it.
  - There are six variables in the above code.
- The value is substituted for the variable when the variable appears on the right hand side of the `=`.
- The value is **assigned to** the variable when the variable name appears on the left hand side of the `=`.

More on Variable Assignment

- The operator `=` is an assignment of a value (calculated on the right side) to a variable (on the left).
- In the following..

```
>>>  base_area = pi * radius ** 2
```

  Python
  - accesses the values associated with the variables `pi` and `radius`,
  - squares the value associated with `radius` and then multiplies the result by the value associated with the variable `pi`,
  - associates the result with the variable `base_area`
- Later, Python accesses the value of `base_area` when calculating the values to assign to `volume` and `surface_area`.
- Thus, the meaning of `=` in Python is quite different from the meaning of `=` in mathematics.
- The statement

```
>>>  base_area * height = volume
```

  is not legal Python code. Try it!
- It takes a while to get accustomed to the meaning of an assignment statement in Python.

print

- Consider the line

```
>>> print("volume is", volume, ", surface area is", surface_area)
```

- `print` is a Python "function" that combines *strings* (between the quotations) and values of variables, separated by commas, to generate nice output.
- We will play with a number of examples in class to illustrate use of `print`. As the semester progresses we will learn a lot more about it.

Variable Names

- Notice that our example variable names include letters and the `_` (underscore) character.
- Legal variable names in Python must
  - Start with a letter or a `_`, and
  - Be followed by any number of letters, underscores or digits.

  Characters that are none of these, including spaces, signal the end of a variable name.
- Capital letters and small letters are different
- We will look at many examples in class.

Putting Your Code in a File

- So far in today's lecture we have written our code using the *Python Shell*.
  - This sends your Python statements directly to the *interpreter* to execute them
- Now we will switch to writing and saving our code in a file and then sending the file to the Python interpreter to be run.
- We will demonstrate using the surface area and volume calculations from earlier in lecture.
- Almost all code that you write for lecture exercises, labs, and homework assignments will be stored in files.
- You will practice in Lab 1
- Sometimes in class we will still type things directly into the shell. You will know we are doing this when you see `>>>`

Syntax and Semantic Errors

- Python tells us about the errors we make in writing the names of variables and in reversing the left and right side of the `=` operator.
- These are examples of *syntax errors* — errors in the form of the code.

- Programs with syntax errors will not run; the Python interpreter inspects the code and tells us about these errors before it tries to execute them. We can then fix the errors and try again
- More difficult to find and fix are *semantic errors* — errors in the logical meaning of our programs resulting in an incorrect result.
    - We have already seen an example of a semantic error. Can you think where?
    - Throughout the semester we will discuss strategies for finding and fixing semantic errors.

Python Keywords

- All variable names that follow the above rules are legal Python names *except* for a set of "keywords" that have special meaning to Python.
- Keywords allow us to write more complicated operations — involving logic and repetition — than just calculating.
- You can get a list of Python keywords by typing into the shell

```
>>> import keyword
>>> print(keyword.kwlist)
```

- Over the next few lectures, we will soon understand the detailed meaning of the . in the above statement.

Do Variables Exist Before They Are Assigned a Value?

- Suppose we forgot to assign `pi` a value? What would happen?
    - Try it out!
- Variables do not exist until they are assigned a value.
- This is a simple form of semantic error.

Example to Consider

1. Create 2 invalid variable names and 4 valid variable names from the `_` character, the digit `0`, and the letter `a`.

Mixed Operators

- Assignments of the form

```
>>> i = i + 1
```

are commonly seen in Python. We will take a careful look at what is happening here.

- Python contains a short-hand for these:

```
>>> i += 1
```

These two statements are exactly equivalent.

- Other mixed operators include

```
-=     *=    /=
```

but `+=` is used most commonly for reasons that will gradually become clear over the first half of the semester.

Terminology: Expressions

- Expressions are formed from combinations of values, variables and operators.
- In the examples we've seen so far, expressions are on the right-hand side of an assignment statement, as in:

```
>>> surface_area = 2 * base_area + 2 * pi * radius * height
```

Part 2 Practice Problems

1. Which of the following are legal Python variable names?

```
import
56abc
abc56
car-talk
car_talk
car talk
```

2. Which of these lines of code contain syntax errors? Once you fix the syntax errors, the program (assume this has been typed into a file and run in the Spyder IDE) will still not correctly print the area of a circle with radius 6.5. What two more changes are needed to fixed these errors?

```
pi = 21 // 7
area = pi * r * r
r = 6.5
r + 5 = r_new
print(area)
```

3. Assuming you start with $100 and earn 5% interest each year, how much will you have at the end of one year, two years and three years? Write Python expressions to calculate these, using variables as appropriate. We will write the solution into a file and run the file using the interpreter.
4. What is the output of the following Python code (when typed into a file and run in the interpreter)? Try to figure it out by hand before typing the statements into a file and running in the Python interpreter.

```
x = 12
y = 7.4
x -= y
print(x, y)
y = y-x +7
z = 1
x *= 2 + z
print(x, y)
x += x*y
print(x, y)
```

Summary — Important Points to Remember

- Expressions are formed from combinations of values, variables and operators
- Values in Python are one of several different types — integers, floats, and strings for now.
- Variables are Python's form of memory
- Python keywords cannot be used as variables.
- `=` is Python's means of assigning a value to a variable
- Variables do not exist in Python until they are given a value
- Make sure you have the precedence correct in your Python expressions.