Lecture 4 — Using functions and modules

Reading

- Material for this lecture is drawn from Sections 3.1, 6.1 and 7.1-7.3 of *Practical Programming*.
- Topics that we will discuss include:
  - Python functions for different data types.
  - String functions and calling functions on objects
  - Using modules provided with Python
- We will revisit all these concepts several times throughout the semester.

What have we learned so far?

- So far, we have learned about three basic data types: integer, float and strings.
- We also learned some valuable functions that operate on strings (`len`) and that convert between data types (`int`, `str`, `float`)

```
>>> name = "Pickle Rick"
>>> len(name)
11
```

- The functions that Python provides are called *built-in* functions.
- We will see examples of these functions and experiment with their use in this class.

How about numerical functions?

- Many numerical functions also exist. Let us experiment with some of these first. You should make a note of what they do.

    - `abs()`
    - `pow()`
    - `int()`
    - `float()`
    - `round()`
    - `max()`
    - `min()`
- We will look carefully in class at how these work.

Objects and Methods

- All variables in Python are objects.
- Objects are abstractions:
    - Each object defines an organization and structure to the data they store.
    - They have operations/functions — we call them **methods** — that we can apply to access and manipulate this data.
    - We don't think about how they are implemented; instead we just think about how to use them. This is why they are called abstractions.
- Methods associate with objects often use a function call syntax of the form

```
variable.method(arguments)
```

  For example:

```
>>> b = 'good morning'
>>> b.find('o', 3)
```

  This also works on particular values instead of variables:

```
value.method(arguments)
```

  as in

```
>>> 'good morning'.find('o', 3)
```

- You can see all the methods that apply to an object type with help as well. Try:

```
>>> help(str)
```

String Methods

- Here are a few more (of many) string methods:

```
>>> name = "Neil Degrasse Tyson"
>>> name.lower()
'neil degrasse tyson'
>>> lowername = name.lower()
>>> lowername.upper()
'NEIL DEGRASSE TYSON'
>>> lowername.capitalize()
'Neil degrasse tyson'
>>> lowername.title()
'Neil Degrasse Tyson'
>>> "abracadabra".replace("br", "dr")
'adracadadra'
>>> "abracadabra".replace("a", "")
'brcdbr'
>>> "Neil Degrasse Tyson".find(" ")
4
>>> "Neil Degrasse Tyson".find("a")
9
>>> "Neil Degrasse Tyson".find("x")
-1
>>> "Monty Python".count("o")
2
>>> "aaabbbfsassassaaaa".strip("a")
'bbbfsassass'
```

- As described above, all of these are called in the form of `object.method(arguments)`, where `object` is either a string variable or a string value.
- Not all functions on objects are called this way. Some are called using more of a function form, while others are called as operators.

```
>>> episode = "Cheese Shop"
>>> episode.lower()
'cheese shop'
>>> len(episode)
11
>>> episode + "!"
'Cheese Shop!'
```

  - We will see the reason for the differences later in the semester.
- Note of caution: none of these functions change the variable that they are applied to.

Practice Problems (1)

1. Write code that takes a string in a variable called `phrase` and prints the string with all vowels removed.
2. Create a string and assign it to a variable called `name`. Write code to create a new string that repeats each letter `a` in `name` as many times as `a` appears in `name` (assume the word is all lower case).

   For example,

```
>>> name = "amos eaton"
## your code goes here



>>> name
'aamos eaaton'
```

3. Given a string in a variable called `name`, switch all letters `a` and `e` (only lowercase versions). Assume the variable contains only letters and spaces.

   Hint: first replace each 'a' with '1'.

```
>>> name = "Rensselaer Polytechnic Institute"
## your code goes here



>>> name
'Ranssalear Polytachnic Instituta'
```

String Format Method

- The `format()` method provides a nice way to produce clean looking output.
- For example, consider the code

```
>>> pi = 3.14159
>>> r = 2.5
>>> h = 10**0.5
>>> volume = pi * r**2 * h
>>> print('A cylinder of radius', r, 'and height', h, 'has volume', volume)
A cylinder of radius 2.5 and height 3.1622776601683795 has volume 62.09112421505237
```

- Now look at what we can do with the `format()` method:

```
>>> out_string = 'A cylinder of radius {0:.2f} and height {1:.2f} has volume
{2:.2f}'.format(r, h, volume)
>>> print(out_string)
A cylinder of radius 2.50 and height 3.16 has volume 62.09
```

- Method `format()` replaces the substrings between { } with values from the argument list.
  - {0:.2f} means argument 0, will be formatted as a float with 2 digits shown to the right of the decimal place.
    - Notice it applies rounding.
  - We can leave off the 0, the 1, and the 2 from before the : unless we want to change the order of the output.
  - We can leave off the :.2f if we want to accept print's normal formatting on float outputs.
- There are many variations on this and we will see quite a few as we progress through the semester.

Built-In Functions

- All the functions we have seen so far are *built-in* to the core Python. It means that these functions are available when you start Python.
- Type

```
>>> help(__builtins__)
```

to see the full list.

Modules

- Now we will begin to look at using functions that are not built into the core of Python but rather imported as **modules**.
- Modules are collections of functions and constants that provide additional power to Python programs.
- Some modules come with Python, but are not loaded automatically. For example the `math` module.
- Other modules need to be installed first. When we installed software in Lab 0, we installed a library called `pillow` that has a number of image manipulation modules.
- To use a function in a module, first you must load it into your program using `import`. Let's see the `math` module:

```
>>> import math
>>> math.sqrt(5)
2.2360679774997898
>>> math.trunc(4.5)
4
>>> math.ceil(4.5)
5.0
>>> math.log(1024,2)
10.0
>>> math.pi
3.141592653589793
```

- We can get an explanation of what functions and variables are provided in a module using the `help` function

```
>>> import math
>>> help(math)
```

Different Ways of Importing

- The way you import a module determines what syntax you need to use the contents of the module in your program.
- We can import only a selection of functions and variables:

```
>>> from math import sqrt,pi
>>> pi
3.141592653589793
>>> sqrt(4)
2.0
```

- Or we can give a new name to the module within our program:

```
>>> import math as m
>>> m.pi
3.141592653589793
>>> m.sqrt(4)
2.0
```

- Both of these methods help us distinguish between the function `sqrt` and the data `pi` defined in the math module from a function with the same name (if we had one) in our program.
- We can also do this (which is NOT recommended!):

```
>>> from math import *
```

Now, there is no name difference between the math module functions and ours. Since this leads to confusion when the same name appears in two different modules it is almost always avoided.

Program Structure

- We have now seen several components of a program: `import`, comments, and our own code, including input, computation and output statements. We will add more components, such as our own functions, as we proceed through the semester.
- You should organize these components in your program files to make it easy to see the flow of the program
- We will use the following convention to order the program components:
  - an initial comment explaining the purpose of the program,
  - all `import` statements,
  - then all variables and input commands,
  - then all computation,
  - finally all output.

Putting It All Together

- In the rest of this class we will write a program that first asks the user for a name, then asks for the radius and height of a cylinder, and finally prints the surface area and volume of the cylinder, nicely formatted.

Practice Problems (2)

1. The `math` module contains the constant `e` as well as `pi`. Write code that prints these values accurate to 3 decimal places and then write code that computes and outputs

   $$\pi^e$$

   and

   $$e^\pi$$

   both accurate to 2 decimal places.
2. Write a short program to ask the user to input height values (in cm) three times. After reading these values (as integers), the program should output the largest, the smallest and the average of the height values.

3. What happens when we type

```
import math
math.pi = 3
```

and then use `math.pi`?

Summary

- Python provides many functions that perform useful operations on strings, integers and floats.
- Some of these functions are *built in* while others are organized into modules.
- Be aware of the differences between how functions are called. You must remember them to call them correctly.
    - Functions that require dot notation, applying the function to an object (or a variable containing an object):

```
>>> "abc".upper()
'ABC'
```

    - Functions that are called with arguments (no dot notation):

```
>>> x = -4.6
>>> abs(x)
4.6
>>> round(x)
-5
```

      Note that these functions are actually aliases. The same function also exists in dot notation.

```
>>> x = -4.6
>>> x.__abs__()
4.6
>>> x.__round__()
-5
>>> 4.6.__round__()
5
>>> (-1).__abs__()
1
```

- After a module is imported, the functions in the module can be used by a call of the form:

```
module_name.function_name(arguments)
```

- You can see the details of a function by:

```
>>> help(module_name.function_name)
```

- Python has many modules that make it easy to do complicated tasks. If you do not believe it, try typing:

```
>>> import antigravity
```