Chapter 2 Variables and Simple Data Types

- 2.1 What Really Happens When You Run hello_world.py
- 2.2 Variables
- 2.3 Strings
- 2.4 Numbers
- 2.5 Comments
- 2.6 The Zen of Python

2.1 What Really Happens When You Run hello_world.py

- **hello_world.py**

```
print("Hello Python world!")
```

When you run this code, you should see this output:

```
Hello Python world!
```

2.2 Variables

- **hello_world.py**

```
message = "Hello Python world!"
print(message)
```

Run this program to see what happens. You should see the same output you saw previously:

```
Hello Python world!
```

- **Let's expand on this program by modifying hello_world.py to print a second message.**

```
message = "Hello Python world!"
print(message)
message = "Hello Python Crash Course world!"
print(message)
```

Now when you run *hello_world.py*, you should see two lines of output:

```
Hello Python world!
Hello Python Crash Course world!
```

2.2.1 Naming and Using Variables

- Variable names can contain only letters, numbers, and underscores
- Spaces are not allowed in variable names, but underscores can be used to separate words in variable names
- Avoid using Python keywords and function names as variable names
- Variable names should be short but descriptive
- Be careful when using the lowercase letter *l* and the uppercase letter *O* because they could be confused with the numbers *1* and *0*

2.2.2 Avoiding Name Errors When Using Variables

Demonstrate an example

2.3 String

- A *string* is simply a series of characters. Anything inside quotes is considered a string in Python, and you can use single or double quotes around your strings like this:

```
"This is a string."
'This is also a string.'
```

- This flexibility allows you to use quotes and apostrophes within your strings:

```
'I told my friend, "Python is my favorite language!"'

"The language 'Python' is named after Monty Python, not
 the snake."

"One of Python's strengths is its diverse and supportive
 community."
```

2.3.1 Changing Case in a String with Methods

- One of the simplest tasks you can do with strings is change the case of the words in a string.
- name.py

```
name = "ada lovelace"
print(name.title())
```

Save this file as *name.py* , and then run it. You should see this output:

```
Ada Lovelace
```

- You can change a string to all uppercase or all lowercase letters like this:

```
name = "Ada Lovelace"
print(name.upper())
print(name.lower())
```

This will display the following:

```
ADA LOVELACE
ada lovelace
```

2.3.2 Combining or Concatenating Strings

- You might want to store a first name and a last name in separate variables, and then combine them when you want to display someone's full name:

```
first_name = "ada"
last_name = "lovelace"
full_name = first_name + " " + last_name
print(full_name)
```

The result is as follows:

```
ada lovelace
```

- This method of combining strings is called *concatenation*.

- How to return a simple but nicely formatted greeting by using concatenation.

```
first_name = "ada"
last_name = "lovelace"
full_name = first_name + " " + last_name
print("Hello, " + full_name.title() + "!")
```

The above two codes show the results:

```
Hello, Ada Lovelace!
```

2.3.3 Adding Whitespace to Strings with Tabs or Newlines

- In programming, *whitespace* refers to any nonprinting character, such as spaces, tabs, and end-of-line symbols. You can use whitespace to organize your output so it's easier for users to read.
- To add a tab to your text, use the character combination \t as shown at❶:

```
>>> print("Python")
Python
❶ >>> print("\tPython")
      Python
```

- To add a newline in a string, use the character combination \n :

```
>>> print("Languages:\nPython\nC\nJavaScript")
Languages:
Python
C
JavaScript
```

- You can also combine tabs and newlines in a single string. The string "\n\t" tells Python to move to a new line, and start the next line with a tab. The code example is as follows:

```
>>> print("Languages:\n\tPython\n\tC\n\tJavaScript")
Languages:
    Python
    C
    JavaScript
```

2.3.4 Stripping Whitespace

- Python can look for extra whitespace on the right and left sides of a string. To ensure that no whitespace exists at the right end of a string, use the rstrip() method.

```
❶ >>> favorite_language = 'python '
❷ >>> favorite_language
  'python '
❸ >>> favorite_language.rstrip()
  'python'
❹ >>> favorite_language
  'python '
```

- To remove the whitespace from the string permanently, you have to store the stripped value back into the variable:

```
>>> favorite_language = 'python '
❶ >>> favorite_language = favorite_language.rstrip()
>>> favorite_language
'python'
```

- You can also strip whitespace from the left side of a string using the lstrip() method or strip whitespace from both sides at once using strip() :

```
❶ >>> favorite_language = ' python '
❷ >>> favorite_language.rstrip()
  ' python'
❸ >>> favorite_language.lstrip()
  'python '
❹ >>> favorite_language.strip()
  'python'
```

2.3.5 Avoiding Syntax Errors with Strings

- Here's how to use single and double quotes correctly. Save this program as *apostrophe.py* and then run it:

```
message ="One of Python's strengths is its diverse
community."
print(message)
```

- The apostrophe appears inside a set of double quotes, so the Python interpreter has no trouble reading the string correctly:

```
One of Python's strengths is its diverse community.
```

- However, if you use single quotes, Python can't identify where the string should end:

```
message ='One of Python's strengths is its diverse
------community.'
print(message)
```

- You'll see the following output:

```
File "apostrophe.py", line 1
message ='One of Python's strengths is its diverse
------community.'
                        ^ ❶
SyntaxError: invalid syntax
```

2.3.6 Printing in Python 2

- The print statement has a slightly different syntax in Python 2:

```
>>> python2.7
>>> print "Hello Python 2.7 world!"
Hello Python 2.7 world!
```

- Parentheses are not needed around the phrase you want to print in Python 2.

2.4 Numbers

- Numbers are used quite often in programming to keep score in games, represent data in visualizations, store information in web applications, and so on. Python treats numbers in several different ways, depending on how they are being used. Let’s first look at how Python manages integers, because they are the simplest to work with.

2.4.1 Integers

- You can add (+), subtract (-), multiply (*), and divide (/) integers in Python.

```
>>>2 + 3
5
>>>3 - 2
1
>>>2 * 3
6
>>>3 / 2
1.5
```

- In a terminal session, Python simply returns the result of the operation. Python uses two multiplication symbols to represent exponents:

```
>>> 3 ** 2
9
>>> 3 ** 3
27
>>> 10 ** 6
1000000
```

- Python supports the order of operations too, so you can use multiple operations in one expression. You can also use parentheses to modify the order of operations so Python can evaluate your expression in the order you specify. For example:

```
>>> 2 + 3*4
14
>>> (2 + 3) * 4
20
```

2.4.2 Floats

- Python calls any number with a decimal point a *float*.
- For the most part, you can use decimals without worrying about how they behave. Simply enter the numbers you want to use, and Python will most likely do what you expect:

```
>>>0.1 + 0.1
0.2
>>>0.2 + 0.2
0.4
>>>2 * 0.1
0.2
>>>2 * 0.2
0.4
```

- But be aware that you can sometimes get an arbitrary number of decimal places in your answer:

```
>>> 0.2 + 0.1
0.30000000000000004
>>> 3 * 0.1
0.30000000000000004
```

2.4.3 Avoiding Type Errors with the str() Function

- Often, you'll want to use a variable's value within a message. For example, say you want to wish someone a happy birthday. You might write code like this:

```
age = 23
message = "Happy " + age + "rd Birthday!"
print(message)
```

- You might expect this code to print the simple birthday greeting, Happy 23rd birthday! But if you run this code, you'll see that it generates an error:

```
  Traceback (most recent call last):
  File "birthday.py", line 2, in <module>
  message = "Happy " + age + "rd Birthday!"
❶TypeError:Can't convert 'int' object to str implicitly
```

- This is a type error. It means Python can’t recognize the kind of information you’re using. In this example Python sees at❶ that you’re using a variable that has an integer value (int), but it’s not sure how to interpret that value. Python knows that the variable could represent either the numerical value 23 or the characters 2 and 3. When you use integers within strings like this, you need to specify explicitly that you want Python to use the integer as a string of characters. You can do this by wrapping the variable in the str() function, which tells Python to represent non-string values as strings:

```
age = 23
message = "Happy " + str(age) + "rd Birthday!"
print(message)
```

- Python now knows that you want to convert the numerical value 23 to a string and display the characters 2 and 3 as part of the birthday message. Now you get the message you were expecting, without any errors:

```
Happy 23rd Birthday!
```

2.4.4 Integers in Python 2

- Python 2 returns a slightly different result when you divide two integers:

```
>>> python2.7
>>> 3 / 2
    1
```

- Instead of 1.5 , Python returns 1 . Division of integers in Python 2 results in an integer with the remainder truncated. Note that the result is not a rounded integer; the remainder is simply omitted.

- To avoid this behavior in Python 2, make sure that at least one of the numbers is a float. By doing so, the result will be a float as well:

```
>>>3 / 2
1
>>>3.0 / 2
1.5
>>>3 / 2.0
1.5
>>>3.0 / 2.0
1.5
```

2.5 Comments

- Comments are an extremely useful feature in most programming languages. Everything you've written in your programs so far is Python code. As your programs become longer and more complicated, you should add notes within your programs that describe your overall approach to the problem you're solving. A comment allows you to write notes in English within your programs.

- In Python, the hash mark (#) indicates a comment. Anything following a hash mark in your code is ignored by the Python interpreter. For example:

```
# Say hello to everyone.
print("Hello Python people!")
```

Python ignores the first line and executes the second line.

```
Hello Python people!
```

2.6 The Zen of Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.

- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

2.7 Summary

- In this chapter you learned to work with variables. You learned to use descriptive variable names and how to resolve name errors and syntax errors when they arise. You learned what strings are and how to display strings using lowercase, upper-case, and titlecase. You started using whitespace to organize output neatly, and you learned to strip unneeded whitespace from different parts of a string. You started working with integers and floats, and you read about some unexpected behavior to watch out for when working with numerical data. You also learned to write explanatory comments to make your code easier for you and others to read. Finally, you read about the philosophy of keeping your code as simple as possible, whenever possible.
- In Chapter 3 you'll learn to store collections of information in variables called lists. You'll learn to work through a list, manipulating any information in that list.