CSC 1010 Lecture 2

What do we know so far?

- Class – lecture, lab, Rephactor, Quick Checks, R&R
- Program to solve problems, make computers useful
- User vs. Programmer – think like both!
- Program – sequence of instructions
- Python is 3rd most popular language, core principles
- Algorithm is step-by-step procedure to do something
- syntax, runtime, & logic errors, testing & debugging
- hardware vs. software
- flow – step-by-step, function call, conditional, loop
- IDLE shell >>> or editor for saving and running
- Lab 1 – visual programming, install Python, Hello World

Interpreting vs. Compiling

An interpreter goes one instruction at a time, translating into machine code and executing each on the machine (Python)

A compiler translates all instructions first into machine code, then executing all of them on the machine (Python)

Python Standard Library

The Python Standard Library is a toolbox of ready-to-use, built-in components for your program.

- built-in functions - frequently used functions such as print
- built-in types – numbers, sequences, and more
- built-in constants – fixed values like True, False and math.pi
- built-in exceptions - represent errors and warnings

Some modules require the import statement to use them.

```python
import math
print('The value of PI is', math.pi)
```

The value of PI is 3.141592653589793

Variables

A variable is a name used to refer to a value stored in the computer’s memory

```python
name value
total = 500
count = 7.5
```

Print out the value inside a variable like this:

```python
print(total)
print("The value of count is", count)
```
Assignment Statements

An assignment statement changes the value of a variable
The assignment operator is the = sign

```
total = 55
```

- The expression on the right is evaluated and the result is stored in the variable on the left
- The value that was in total is overwritten

Numeric Expressions

An expression is a combination of one or more operators and operands
Numeric expressions compute numeric results and make use of the arithmetic operators:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Floor Division (//)
- Modulus (%)
- Exponentiation (**)

Division and Remainder

The result of division (/) is always floating point

```
14 / 3 equals 4.666666666666667
8 / 16 equals 0.5
```

The modulus operator (%) returns the remainder after dividing the second operand into the first

```
14 % 3 equals 2
8 % 12.0 equals 8.0
```

If either operand of modulus is floating point, the result is floating point, too

Floor Division and Exponentiation

If both operands to the floor division operator (//) are integers, the result is integer (the fractional part isn’t shown)

```
14 // 3 equals 4
17.0 // 5 equals 3.0
```

The exponentiation operator (**) returns the result of raising the first number to the power of the second

```
2 ** 3.0 equals 8.0
5 ** 2 equals 25.0
```

Operator Precedence

Operators can be combined into complex expressions
```
result = total + count / max - offset
```

Operators have a well-defined precedence which determines the order in which they are evaluated, which is:

1. exponentiation
2. multiplication, division, floor division, modulus
3. addition, subtraction

Same precedence? Go left to right
Use parentheses to force precedence

Operator Precedence

What is the order of evaluation in the following expressions?

```
a + b + c + d + e
1 2 3 4
```

```
a + b * c - d / e
3 1 4 2
```

```
a // b ** c - d % e
2 1 4 3
```

```
a / (b * (c + (d - e)))
4 3 2 1
```
The print Function

The `print` function writes text output to the console window.
```
pint('Hey there, World! Sup?')
Hey there, World! Sup!
```

Multiple arguments print on the same line with spaces between
```
height = 24.56832
print('The height is', height)
The height is 24.56832
```

Optional print arguments

The `end` argument overrides what prints at the end, which is normally a "newline" (next print goes on the next line).
```
print('Hello ', end='')
print('World!')
Hello World!
```

The `sep` argument overrides what goes between values, normally a space.
```
print('one', 'two', 'three', 'four', sep='$$')
one$$$two$$$three$$$four
```

Strings

A character string is a sequence of characters, surrounding by single or double quotes
```
killer = 'Lee Harvey Oswald'
victim = "John F. Kennedy"
```

To include one type of quote inside a string, use the other type of quotes on the ends
```
sentence = "Kennedy's killer is Oswald"
print('There are many "assassination conspiracies" out there')
```

String Concatenation

String concatenation allows two or more strings to be joined using the `+` operator.
```
value = "The killer of + victim + is + killer"
```

To concatenate numbers to strings, use the `str` function around them or you'll get an error message.
```
name = 'Robert Wadlow'
feet = 8
inches = 11
print('name' + 'feet' + 'inches' + 'tall')
Robert Wadlow was 8 feet 11 inches tall.
Long Strings
Character strings cannot be broken across lines:

```python
print('This is a very long string but it is quite possible there are even longer strings!')
```

You can use string concatenation for long strings

```python
print('This is a very long string but it is quite possible there are even longer strings!')
```

You can also surround long strings with triple quotes

```python
print('''This is a very long string but it is quite possible there are even longer strings!''')
```

String Indexes
Each character in a string has a position or index

<table>
<thead>
<tr>
<th>Index</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>R</td>
</tr>
<tr>
<td>1</td>
<td>e</td>
</tr>
<tr>
<td>2</td>
<td>p</td>
</tr>
<tr>
<td>3</td>
<td>h</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
</tr>
<tr>
<td>6</td>
<td>t</td>
</tr>
<tr>
<td>7</td>
<td>o</td>
</tr>
<tr>
<td>8</td>
<td>r</td>
</tr>
<tr>
<td>9</td>
<td>e</td>
</tr>
<tr>
<td>10</td>
<td>f</td>
</tr>
<tr>
<td>11</td>
<td>a</td>
</tr>
<tr>
<td>12</td>
<td>c</td>
</tr>
<tr>
<td>13</td>
<td>t</td>
</tr>
<tr>
<td>14</td>
<td>o</td>
</tr>
<tr>
<td>15</td>
<td>n</td>
</tr>
</tbody>
</table>

You can access an individual character using its index and the index operator (square brackets []). Suppose the above string is in a variable called `title`:

```python
print('Character 6 in title is ' + title[6])
```

Character 6 in title is t

String Length
You can determine the length of a string using `len`

```python
title = 'Rephactor Python'
print(len(title))
```

16

If you compare the length to the last index, you'll see it is different. That's because indexes start counting at 0.

<table>
<thead>
<tr>
<th>Index</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>R</td>
</tr>
<tr>
<td>1</td>
<td>e</td>
</tr>
<tr>
<td>2</td>
<td>p</td>
</tr>
<tr>
<td>3</td>
<td>h</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
</tr>
<tr>
<td>6</td>
<td>t</td>
</tr>
<tr>
<td>7</td>
<td>o</td>
</tr>
<tr>
<td>8</td>
<td>r</td>
</tr>
<tr>
<td>9</td>
<td>e</td>
</tr>
<tr>
<td>10</td>
<td>f</td>
</tr>
<tr>
<td>11</td>
<td>a</td>
</tr>
<tr>
<td>12</td>
<td>c</td>
</tr>
<tr>
<td>13</td>
<td>t</td>
</tr>
<tr>
<td>14</td>
<td>o</td>
</tr>
<tr>
<td>15</td>
<td>n</td>
</tr>
</tbody>
</table>

Iterating Through a String
You can step through the characters or iterate a string like this

```python
language = 'Python'
for letter in language:
    print(letter)
```

P
y
t
h
o
n
This iteration uses a for loop

String Containment
To see if one string is contained inside another string, use the in operator

```python
title = 'Rephactor Python'
if 'actor' in title:
    print('Found it!')
```

Found it!

Repeat a String
To repeat any string some number of times, use the repetition operator (*)

```python
books = 'Python' * 7
print(books)
```

PythonPythonPythonPythonPythonPythonPython

The order of the two doesn't matter, but there has to be a string and an integer

```python
hashtags = 15 * '#'
print(hashtags)
```

###############
Comments explain a program’s purpose and processing. They are intended for the human reader—they have no effect on a program.

A Python comment begins with a `#` and continues until the end of the line.

```
# This is a comment
```

It might be put on the end of a line of code.

```
balance = balance - fees  # deduct monthly fees
```

A block comment has as many lines of comments as needed.

```
# Function to print a quote from The Simpsons episode titled Bart vs. Thanksgiving.
# s2, ep7
# def print_simpsons_quote():
#     print("Operator, what's the number for 9-1-1?")
#     print("   - Homer Simpson")
# Main program
print_simpsons_quote()
```

```
Operator, what's the number for 9-1-1!
   - Homer Simpson
```