Chapter 2: Variables, Operations, and Strings

CSCI-UA 0002 – Introduction to Computer Programming

Mr. Joel Kemp
Agenda

- Our First Program
- Variables
- Expressions
- Assignment
- Datatypes
- Type Casting
- Operators
- Console I/O
Our First Program

What does this program do?

greeting = "Hello there"
first_name = "Joel"
middle_name = "Edward"
last_name = "Kemp"

print(greeting, first_name, middle_name, last_name)

The output:
Hello there Joel Edward Kemp

But what are greeting, first_name, middle_name, and last_name?
Variables

• **Definition:** name/identifier given to some unknown data value stored in a computer’s memory.

• **Format Rules:**
  - Start with any letter (or an underscore) but no numbers!
    
    g, guess, var, avg1, ycoord, point2, name

  - No spaces or other punctuation allowed!
    
    - Use an underscore instead! (“Snake Case”)
      
      first_name, celsius_temp, wind_speed

  - No *reserved words*!
    
    - **Reserved word:** an identifier that is part of the Python language.

  - Identifiers should be descriptive.
# Reserved Words

<table>
<thead>
<tr>
<th>and</th>
<th>del</th>
<th>from</th>
<th>not</th>
<th>while</th>
</tr>
</thead>
<tbody>
<tr>
<td>as</td>
<td>elif</td>
<td>global</td>
<td>or</td>
<td>with</td>
</tr>
<tr>
<td>assert</td>
<td>else</td>
<td>if</td>
<td>pass</td>
<td>yield</td>
</tr>
<tr>
<td>break</td>
<td>except</td>
<td>import</td>
<td>print</td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>exec</td>
<td>in</td>
<td>raise</td>
<td></td>
</tr>
<tr>
<td>continue</td>
<td>finally</td>
<td>is</td>
<td>return</td>
<td></td>
</tr>
<tr>
<td>def</td>
<td>for</td>
<td>lambda</td>
<td>try</td>
<td></td>
</tr>
</tbody>
</table>
Variables cont.

• What can you do with a variable?
  – **Assign** a value (write)
    
    ```
    x_coord = -3
    price = 5.50
    is_hungry = True
    name = “Joel Kemp”
    ```
  – **Retrieve** a value (read)
    
    ```
    tip = 0.15 * bill_amount
    last_name = first_name  # It can happen!
    print(tip)
    ```

• What’s going on with this statement?
  
  ```
  tip = 0.15 * bill_amount
  ```

Hmm, there are a few strange concepts that need clarification.
Expressions

• What are they?
  – *Statements* that contain any number of variables, constants, and operators.

  • **Statement**: a line of Python code delimited by a carriage return or semicolon.
    – Semicolon is used to put more than one statement on a line.
      » Used as a statement delimiter in other languages.

• Examples:

  ```python
  tip_amount = 0.15 * bill_amount
  num_squared = x * x
  firstName = "Joel"
  True
  print(firstName)
  tip_percent = 0.15
  ```
Assignment Statements

• The most common type of programming statement!
• How do we read/evaluate an assignment?

```python
# Initialization
tip_percent = 0.15
bill_amount = 100
tip = tip_percent * bill_amount
```

– The right hand side is evaluated and the result is stored in the left hand side!

– The `=` sign is known as the **assignment operator**.
• Note: variables on the RHS need to be initialized!
• Python also lets you do **multiple initialization**!

```python
x, y = 5, "Joel"  # Equal to x = 5  y = "Joel"
```
Datatypes

• We can store different types of data in variables!
  – **Integers**: whole numbers like 0, -2, 100
  – **Floats**: fractional numbers like 0.0001, 0.5, 3.5, -20.1
  – **Booleans**: truth values `True` or `False`
  – **Strings**: groups of characters “Joel”, “Kemp”, “Joel Kemp”

• More advanced types:
  – **Objects**: custom datatypes composed of simpler types
  – **Collections**: allow a variable to hold more than one value at a time.
  – **File Handles**: allow variables to manipulate text files.

Refer to: datatypes.py
Conversions

• What if I need to convert an integer to a float?
  – Also known as **type casting** from integers to floats.

• Python does some conversions automatically!

```python
tip = 0.15 * 100
```

– 0.15 is a float, but 100 is an integer
– What’s the datatype of `tip`?
– Integers are automatically converted to floats!

  • 100 becomes 100.0
  • 0.15 * 100.0 yields 15.0
  • 15.0 is stored in the variable `tip`
  • Thus, `tip` has the float datatype.
  • This is an **implicit** conversion.

Refer to: `conversions.py`
OPERATORS
Arithmetic Operators

- Programming languages let you mix and match arithmetic operations!
  - Addition: $+$
  - Subtraction: $-$
  - Multiplication: $\ast$
  - Division: $/$ and $//$
  - Remainder: $\%$
  - Exponentiation: $\ast\ast$

- Precedence: from highest to lowest
  
  $( ) \ast\ast \div \% \div \slash / \ast \plus \minus$
The odd one!

- Integer division vs. Float Division
  - **Float Division** (/) yields the fractional results that you’d expect:
    - \( 1 \div 2 = 0.5 \)
    - \( 3 \div 2 = 1.5 \)
  - **Integer division** drops the fractional part!
    - \( 1 \div 2 = 0 \) #0 instead of 0.5
    - \( 3 \div 2 = 1 \) #1 instead of 1.5
Another Odd One!

• We can compute the remainder of a division using the **modulus operator** %

  1 % 2 = 1
  2 % 4 = 2
  4 % 2 = 0
  10 % 200 = 10

– When do we use it?
  • Computing prime numbers
  • Binary arithmetic
  • Random number generation
Combined Operators

• We know how to do simple arithmetic operations.
• We can combine operations acting on a variable:
  – If we have these types of statements:
    
    \[
    \begin{align*}
    x &= x + 10 & \text{//Add 10 to x and store the result back in x} \\
    y &= y - 5 \\
    x &= x / 2 \\
    y &= y * 2 \\
    z &= z + x + y
    \end{align*}
    \]

  – We can rewrite these statements using combined operators:
    
    \[
    \begin{align*}
    x &= x + 10 \\
    y &= y - 5 \\
    x &= x / 2 \\
    y &= y * 2 \\
    z &= z + x + y
    \end{align*}
    \]
CONSOLE I/O
Console Input / Output

• The “console” is the IDLE shell
  – Could be your terminal or command prompt.
• We can print (output) to the shell using the `print()` function.
• We can read (input) data in from the shell using the `input()` function.
• What’s a function?
  – Function: a group of code with an identifier that performs a certain task.
print()

• What does it do?
  – `print` is a built-in Python function that prints information to the console.

• How do you use it?
  ```python
  print(something)
  ```
  – You pass in data (“arguments”)
  – *Something* could be:
    • variable
    • constant/literal
    • expression
    • function call

Refer to: print.py
• **What does it do?**
  – *input* is a built-in Python function that grabs entered text from the console.
    • It captures input and returns it to your program.
• **How do we use it?**
  ```python
  input(prompt)
  ```
  – *Prompt* could be any message that you’d like to show to your users.
• **What happens next?**
  – The user enters some data and hits enter
  – The information is returned to your program as a **string**.
  – You store that string in a variable and do whatever is necessary!
• **Example:**
  ```python
  name = input("Please enter your name: ")
  ```
• So name now has a string containing what the user gave us.
Explicit Conversions

• What if I wanted to convert a string to a number?
  – Implicit conversions were nice, but a bit limited.
• When would you want to do that?
  – Well `input()` only gives us strings...
  – If the user supplied a number, `input()` would return it to the program as a string.
  – What if I wanted to use that input in a formula?
    • No can do 😞
    • Unless, we have some way of converting a string to a number!

• There are explicit conversions that can be used for certain type casts.
Explicit Conversions (cont.)

• Functions can be used to handle explicit conversions:
  - `float()`
  - `int()`
  - `str()`
  - `bool()`

• These functions take in a single argument of **any type** and return the proper representation of that input.

Refer to: explicitConversions.py
STRINGS
Agenda

• Declaring Strings
• The `len()` function
• Concatenation
String

• What is it?
  – A group of one or more characters.

• How do we use them?
  – Three ways to create strings:
    • Single, double, and triple quotes
  – Triple quotes are used for multi-line strings
  – Single quotes work well until you want to output single quotes 😞
  – Double quotes are the preferred method for creating strings.

• Note: Strings are immutable!
  – Immutable: not able to be changed.
The length function, `len()`, computes the length of a string.

- You pass the string as an argument and `len()` returns an integer count of the characters.
- You can use the returned value wherever an integer could be used.

**Example:**

```python
name = input("Please enter your name: ")
print("Your name contains ", len(name), " letters!")
# OR
num_chars = len(name)
print("Your name contains ", num_chars, " letters!")
```
Concatenation

• You can use the addition operator on strings!
• Huh, how do you add words?
• We’re not adding words, we’re combining them!
• Example:
  ```python
  name = "Joel" + "Kemp"    #What does name contain?
  print(name)             #Output is JoelKemp
  ```
• A variation:
  ```python
  name = "Joel"
  name += "Kemp"        # But strings are immutable
  ```
  – You might think we’re modifying the string “Joel” but we’re simply creating a new string “JoelKemp” and storing it in name.
• When would we use this?
  – Combining string inputs from the user.
  – Combining string portions into longer strings.

Refer to: concatenation.py
Exercises

1. Calculate the area and perimeter of a rectangular room, 12.5 meters by 16.7 meters.

2. Write a program to convert temperatures from Fahrenheit to Celsius. The formula for that is:
   \[ C = \frac{5}{9} \times (F - 32) \]
   – Accept the Fahrenheit temperature from the user.
   – Print out the converted temperature!

3. Write a program that figures out how long it will take to get somewhere in a car!
   – The formula (in words) is “travel time equals distance divided by speed.”
   – Make a program to calculate the time it will take to drive a distance (specified by the user) at a speed (also specified by the user) and display the answer.