CSc 110, Autumn 2017

Lecture 5: The for Loop and user input

Adapted from slides by Marty Stepp and Stuart Reges
Declaration and assignment

• **variable declaration and assignment:**
  Sets aside memory for storing a value and stores a value into a variable.
  • Variables must be declared before they can be used.
  • The value can be an expression; the variable stores its result.

• Syntax:

  \[
  \text{name} = \text{expression} \\
  \text{zipcode} = 90210 \\
  \text{myGPA} = 1.0 + 2.25
  \]
Using variables

• Once given a value, a variable can be used in expressions:

\[
x = 3 \quad \# \text{x is 3}
\]
\[
y = 5 \times x - 1 \quad \# \text{now y is 14}
\]

• You can assign a value more than once:

\[
x = 3 \quad \# \text{3 here}
\]
\[
x = 4 + 7 \quad \# \text{now x is 11}
\]
Assignment and algebra

• Assignment uses = , but it is not an algebraic equation.
  • = means, "store the value at right in variable at left"
  • The right side expression is evaluated first, and then its result is stored in the variable at left.

• What happens here?

\[
\begin{array}{c|c}
  x & 5 \\
\end{array}
\]

\[
\begin{align*}
x &= 3 \\
x &= x + 2\end{align*}
\]

# ???
Receipt question

Improve the receipt program using variables.

def main():
    # Calculate total owed, assuming 8% tax / 15% tip
    print("Subtotal:")
    print(38 + 40 + 30)

    print("Tax:")
    print((38 + 40 + 30) * .08)

    print("Tip:")
    print((38 + 40 + 30) * .15)

    print("Total:")
    print(38 + 40 + 30 + (38 + 40 + 30) * .15 + (38 + 40 + 30) * .08)
Getting rid of repetition

- Functions

- Variables

- String Multiplication
  - Allows you to print multiple occurrences of the same string without typing them all out

  ```python
  print("meow" * 3)    # meowmeowmeow
  ```

- What if you want to repeat function calls?
Repetition with \texttt{for} loops

- So far, repeating an action results in redundant code:
  
  ```python
  make_batter()
bake_cookies()
bake_cookies()
bake_cookies()
bake_cookies()
bake_cookies()
bake_cookies()
frost_cookies()
  ```

- Python's \texttt{for loop} statement performs a task many times.
  
  ```python
  mix_batter()
  for i in range(1, 6):   # repeat 5 times
    bake_cookies()
frost_cookies()
  ```
for loop syntax

```python
for variable in range (start, stop):
    statement
    statement
    ...
    statement
```

- Set the variable equal to the start value
- Repeat the following:
  - Check if the variable is less than the stop. If not, stop.
  - Execute the statements.
  - Increase the variable's value by 1.
Control structures

- **Control structure**: a programming construct that affects the flow of a program's execution

- Controlled code may include one or more statements

- The `for` loop is an example of a looping control structure
Repetition over a range

print("1 squared =", 1 * 1)
print("2 squared =", 2 * 2)
print("3 squared =", 3 * 3)
print("4 squared =", 4 * 4)
print("5 squared =", 5 * 5)
print("6 squared =", 6 * 6)

• Intuition: "I want to print a line for each number from 1 to 6"

• The for loop does exactly that!

   for i in range(1, 7):
       print(i, "squared = ", i * i)

   • "For each integer i from 1 through 6, print ..."
Loop walkthrough

```python
for i in range(1, 5):
    print(i, "squared =", i * i)

print("Whoo!")
```

Output:

1  squared = 1
2  squared = 4
3  squared = 9
4  squared = 16
Whoo!
Multi-line loop body

```python
print("+-----+")
for i in range(1, 4):
    print("\" /")
print("/ \")
print("+-----+")
```

• Output:
  +-----+
  \   \   
  \   \   
  \   \   
  \   \   
  +-----+

Expressions for counter

```python
high_temp = 5
for i in range(-3, high_temp // 2 + 1):
    print(i * 1.8 + 32)
```

• Output:
  26.6
  28.4
  30.2
  32.0
  33.8
  35.6
Rocket Exercise

• Write a method that produces the following output:

  T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!
  The end.
print(' ', end='')

- Adding", end=''
  allows you to print without moving to the next line
  - allows you to print partial messages on the same line

high_temp = 5
for i in range(-3, high_temp // 2 + 1):
  print(i * 1.8 + 32, end=' ')

- Output:
  26.6  28.4  30.2  32.0  33.8  35.6

- Either concatenate ‘ ‘ to separate the numbers or set end=' '
Changing step size

• Add a third number to the end of range, this is the step size
  • A negative number will count down instead of up

```python
print("T-minus ")
for i in range(10, 0, -1):
    print(str(i) + ", ", end="")
print("blastoff!")
print("The end.")
```

• Output:
  T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!
The end.
Constants

- **constant**: A fixed value visible to the whole program.
  - value should only be set only at declaration; shouldn't be reassigned

**Syntax:**
- Just like declaring a normal variable:
  ```python
  name = value
  ```
  - name is usually in ALL_UPPER_CASE

**Examples:**
  ```
  DAYS_IN_WEEK = 7
  INTEREST_RATE = 3.5
  SSN = 658234569
  ```
Constants and figures

• Consider the task of drawing the following scalable figure:

```
+\/\//\//\//\//\//\//\//\//\//\//\\+  
   |  |  |  |  |  |  |  |  |  |  |  |  |
   |  |  |  |  |  |  |  |  |  |  |  |  |
   |  |  |  |  |  |  |  |  |  |  |  |  |
   +\//\//\//\//\//\//\//\//\//\//\//\+  
```

Multiples of 5 occur many times

```
+\//\//\//\//\+  
   |  |  |  |
   |  |  |  |
   +\//\//\//\+  
```

The same figure at size 2
Constant tables

\[ \text{SIZE} = \ldots \]

- What equation would cause the code to print:
  
  \[
  2 \ 7 \ 12 \ 17 \ 22
  \]

- To see patterns, make a table of \text{SIZE} and the numbers.
  - Each time \text{SIZE} goes up by 1, the number should go up by 5.
  - But \text{SIZE} \times 5 is too great by 3, so we subtract 3.

<table>
<thead>
<tr>
<th>SIZE</th>
<th>number to print</th>
<th>5 \times \text{SIZE}</th>
<th>5 \times \text{SIZE} - 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td>25</td>
<td>22</td>
</tr>
</tbody>
</table>
Constant tables question

• What equation would cause the code to print:
  17 13 9 5 1

• Let's create the constant table together.
  • Each time \texttt{SIZE} goes up 1, the number printed should ...
  • But this multiple is off by a margin of ...

<table>
<thead>
<tr>
<th>SIZE</th>
<th>number to print</th>
<th>$-4 \times \text{SIZE}$</th>
<th>$-4 \times \text{SIZE} + 21$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>-4</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>-8</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>-12</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>-16</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-20</td>
<td>1</td>
</tr>
</tbody>
</table>
Interactive programs

**interactive program**: Reads input from the console.

- While the program runs, it asks the user to type input.
- The input typed by the user is stored in variables in the code.

- Can be tricky; users are unpredictable and misbehave.
- But interactive programs have more interesting behavior.
input

• **input**: An function that can read input from the user.

• **Using an input object to read console input:**

  ```python
  name = input(prompt)
  ```

  • Example:

  ```python
  name = input("type your name: ")
  ```

  • The variable `name` will store the value the user typed in
def main():
    age = input("How old are you? ")

    years = 65 - age
    print(years, " years until retirement!")

• Console (user input underlined):

    How old are you? 29

    Traceback (most recent call last):
        File "<pyshell#13>", line 1, in <module>
          print(65 - age)
    TypeError: unsupported operand type(s) for -: 'int' and 'str'
```python
def main():
    age = int(input("How old are you? "))
    years = 65 - age
    print(years, "years until retirement!")
```

- Console (user input underlined):
  
  How old are you? 29
  36 years until retirement!