CSc 110, Spring 2017

Lecture 3: Expressions, Variables and Loops
Adapted from slides by Marty Stepp and Stuart Reges

Hackles

By Drake Emko & Jen Brodzik

Preston, do you consider programming more of an art or a science?

Quiet! I'm trying to cut and paste 300 lines of code into 7 different places!

Never mind.

http://hackles.org
Data and expressions
Data types

• Internally, computers store everything as 1s and 0s
  104 → 01101000
  'h' → 01101000
  'hi' → 0110100001101001

• How are 104 and h differentiated?

• type: A category of data values.
  • Constrains the operations that can be performed on data
  Examples: integer, real number, string
### Some Python number types

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integers</td>
<td>42, -3, 0, 92634, 1267650600228229401496703205376</td>
</tr>
<tr>
<td>float</td>
<td>real numbers</td>
<td>3.1, 1.4142135623730951, -0.25</td>
</tr>
</tbody>
</table>
Expressions

• **expression**: A value, or operation that produces a value.
  
  • Examples:
    
    \[
    42 \\
    1 + 4 * 5 \\
    (7 + 2) * 6 / 3 \\
    "Hello, world!"
    \]

  • The simplest expression is a *literal value*.
  • A complex expression can use operators and parentheses.

As a program runs, its expressions are *evaluated* to produce values.

• What value does 42 produce?
• What value does 1+5 produce?
Arithmetic operators

- **operator**: Combines multiple values or expressions.

  + addition
  - subtraction (or negation)
  * multiplication
  / division
  // integer division (a.k.a. leave off any remainder)
  % modulus (a.k.a. remainder)
  ** exponent

- An arithmetic operator can be used with mixed number types

  8 / 5.2 produces 1.5384615384615383
  1 + 3.5 produces 4.5
Integer division with //

• When we divide integers with //, the quotient is also an integer.
  • \( 14 \div 4 \) is 3, not 3.5

\[
\begin{array}{lll}
3 & 4 & 52 \\
4 & 10 & 27 \\
12 & 40 & 135 \\
2 & 5 & 75 \\
\end{array}
\]

• More examples:
  • \( 32 \div 5 \) is 6
  • \( 84 \div 10 \) is 8
  • \( 156 \div 100 \) is 1

What happens when you divide by 0?
Integer remainder with %

• The % operator computes the remainder from integer division.
  • 14 % 4 is 2
  • 218 % 5 is 3

\[
\begin{array}{c}
4 \) 14 \\
\underline{12} \\
\underline{2}
\end{array} \quad \begin{array}{c}
5 \) 218 \\
\underline{20} \\
\underline{18} \\
\underline{3}
\end{array}
\]

Applications of % operator:

• Obtain last digit of a number: 230857 % 10 is 7
• Obtain last 4 digits: 658236489 % 10000 is 6489
• See whether a number is odd: 7 % 2 is 1, 42 % 2 is 0

What is the result?
45 % 6 is 6
2 % 2 is 2
8 % 20 is 8
11 % 0 is 11
Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.
    \[ 1 - 2 - 3 \] is \((1 - 2) - 3\) which is \(-4\)
  - But \(*\) / // % have a higher level of precedence than + -
    \[ 1 + 3 * 4 \] is 13
    \[ 6 + 8 // 2 * 3 \]
    \[ 6 + 4 * 3 \]
    \[ 6 + 12 \] is 18
  - Parentheses can force a certain order of evaluation:
    \((1 + 3) * 4\) is 16
  - Spacing does not affect order of evaluation
    \[ 1 + 3 * 4 - 2 \] is 11

<table>
<thead>
<tr>
<th>Operator precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>,</strong></td>
</tr>
<tr>
<td>/</td>
</tr>
<tr>
<td>+</td>
</tr>
</tbody>
</table>
Precedence examples

\[ 1 \times 2.0 + 3 \times 5 \% 4 \]
\[ 2.0 + 3 \times 5 \% 4 \]
\[ 2.0 + 15 \% 4 \]
\[ 2.0 + 3 \]
\[ 5.0 \]

\[ 1 + 8 \% 3 \times 2 - 9 \]
\[ 1 + 2 \times 2 - 9 \]
\[ 1 + 4 - 9 \]
\[ 5 - 9 \]
\[ -4 \]
Precedence questions

• What values result from the following expressions?

  • 9 \( \div \) 5
  • 695 \( \% \) 20
  • 7 + 6 \( \times \) 5
  • 7 \( \times \) 6 + 5
  • 248 \( \% \) 100 \( \div \) 5
  • 6 \( \times \) 3 – 9 \( \div \) 4
  • (5 – 7) \( \times \) 2 \( \times \times \) 2
  • 6 + (18 \( \% \) (17 – 12))
Operation on strings

• **String concatenation**: `+` operator
  
  "Hello," + " world!" is "Hello, world!"

• Example using print statement
  
  ```python
  print("Hello," + " world!")
  ```
Variables
Receipt example

```python
# 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)

• The subtotal expression (38 + 40 + 30) is repeated
• So many print statements
```
Variables

**Variable**: A named location in the computer's memory that holds a value.

- Variables must be initialized before they can be used.
- The value can be an expression; the variable stores its result.

- Syntax for variable assignment:
  
  ```
  name = expression
  ```

- The rules for `name` are the same as for function names:
  Consist of upper and lower case letters, "_", and digits 0 through 9

- Examples:

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>zipcode</td>
<td>90210</td>
</tr>
<tr>
<td>total</td>
<td>3.25</td>
</tr>
</tbody>
</table>

  • `zipcode = 90210`
  • `total = 1.0 + 2.25`
Using variables

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

```plaintext
x = 3  # x is 3
y = 5 * x  # now y is 15
```

• You can assign a value more than once:

```plaintext
x = 3  # 3 here
x = 4 + 7  # 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?

  x = 3
  x = x + 2  # ???

<table>
<thead>
<tr>
<th>x</th>
<th>5</th>
</tr>
</thead>
</table>

  x = 3
  x = x + 2  # ???
Printing a variable's value

• Use `+ str(value)` to print a string and a variable's value on one line.

  • grade = (95.1 + 71.9 + 82.6) / 3.0
  print("Your grade was " + str(grade))

    students = 11 + 17 + 4 + 19 + 14
  print("There are " + str(students) + " students in the course.")

• Output:

  Your grade was 83.2
  There are 65 students in the course.
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)
def main():
    # Calculate total owed, assuming 8% tax / 15% tip
    subtotal = 38 + 40 + 30  # int
    tax = subtotal * .08     # float
    tip = subtotal * .15     # float
    total = subtotal + tax + tip  # float

    print("Subtotal: " + str(subtotal))
    print("Tax: " + str(tax))
    print("Tip: " + str(tip))
    print("Total: " + str(total))
Repetition with for loops

• So far, repeating an action results in redundant code:

```python
makeBatter()
bakeCookies()
bakeCookies()
bakeCookies()
bakeCookies()
bakeCookies()
bakeCookies()
frostCookies()
```

• Python's for loop statement performs a task many times.

```python
mixBatter()
for i in range(1, 6):  # repeat 5 times
    bakeCookies()
frostCookies()
```
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
for loop syntax

```
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.
Indentation

• Python uses indentation to show that lines of code are inside control structures

• Always use only spaces or only tabs, otherwise you will get very confusing errors!
Repetition over a range

```python
print("1 squared = " + str(1 * 1))
print("2 squared = " + str(2 * 2))
print("3 squared = " + str(3 * 3))
print("4 squared = " + str(4 * 4))
print("5 squared = " + str(5 * 5))
print("6 squared = " + str(6 * 6))
```

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

- The `for` loop does exactly that!

```python
for i in range(1, 7):
    print(str(i) + " squared = " + str(i * i));
```

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

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

print("Whoo!")
```

Output:

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

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

- Output:
  
  +-----+
  \    /\n  \   / \
  \  /  \
  \ /   \
  +-----+

27
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
print(' ', end='')

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

    highTemp = 5
    for i in range(-3, int(highTemp / 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=' '  

29