CSc 110, Spring 2017

Lecture 8: input; if/else

Adapted from slides by Marty Stepp and Stuart Reges

My cat's decision making principle:
Returning a value (review)

```python
def name(parameters):
    statements
    ...
    return expression
```

• When Python reaches a return statement:
  • it evaluates the expression
  • it substitutes the return value in place of the call
  • it goes back to the caller and continues after the function call
Functions that return values (review)

• Consider a function that prints the square of an integer
  
  ```python
  def square(n):
      sq = n * n
      print(sq)
  ```

• Python also allow functions to *return values*
  
  ```python
  def square(n):
      sq = n * n
      return sq
  ```

• The program runs the function, computes the answer, and then "replaces" the function call with its computed result value.

• To see the result, we must print it or store it in a variable.
  
  ```python
  • result = square(8)
  • print(result)  # 64
  ```
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.
input

• **input**: A function that reads input from the user.
• **input**: Returns a value of type string

• Using the input function to read console input:

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

  • Example:
  ```python
  user_name = input("What is your name? ")
  ```

  The string the user types in is assigned to the variable `user_name`
input example

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'
def main():
    age = int(input("How old are you? "))

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

• Console (user input underlined):

    How old are you? 29
    36 years until retirement!
The if/else statement
The `if` statement

**Executes a block of statements only if a test is true**

```python
if (test):
    statement
...
statement
```

• Example:

```python
gpa = float(input("gpa? "))
if (gpa >= 2.0):
    print("Application accepted.")
```
The if/else statement

*Executes one block if a test is true, another if false*

```python
if (test):
    statement(s)
else:
    statement(s)
```

• Example:
  ```python
gpa = float(input("gpa? "))
if (gpa >= 2.0):
    print("Welcome to Mars University!")
else:
    print("Application denied.")
```
Relational Operators

• *if* statements use logical tests.

```python
if (test): ...
```

• **Test** is a *boolean* expression that produces a literal value of True or False

• Tests use *relational operators*  
  
<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equals</td>
<td>1 + 1 == 2</td>
<td>True</td>
</tr>
<tr>
<td>!=</td>
<td>does not equal</td>
<td>3.2 != 2.5</td>
<td>True</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>10 &lt; 5</td>
<td>False</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>10 &gt; 5</td>
<td>True</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>126 &lt;= 100</td>
<td>False</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>5.0 &gt;= 5.0</td>
<td>True</td>
</tr>
</tbody>
</table>

Note the equals "==" !!!
Misuse of `if`

• What's wrong with the following code?

```python
percent = float(input("What percentage did you earn? "))

if (percent >= 90):
    print("You got an A!")

if (percent >= 80):
    print("You got a B!")

if (percent >= 70):
    print("You got a C!")

if (percent >= 60):
    print("You got a D!")

if (percent < 60):
    print("You got an F!")
...```

...
Nested if/else

 Chooses between outcomes using many tests

```python
if (test):
    statement(s)
elif (test):
    statement(s)
else:
    statement(s)
```

- Example:
  ```python
  if (x > 0):
      print("Positive")
  elif (x < 0):
      print("Negative")
  else:
      print("Zero")
  ```
Corrected using nested if/else

• Now only 1 path is taken

percent = float(input("What percentage did you earn? "))

if (percent >= 90):
    print("You got an A!")

elif (percent >= 80):
    print("You got a B!")

elif (percent >= 70):
    print("You got a C!")

elif (percent >= 60):
    print("You got a D!")

else (percent < 60):
    print("You got an F!")
Nested if/else/elif

- If it ends with `else`, exactly one path must be taken.
- If it ends with `if`, the code might not execute any path.

```python
if (test):
    statement(s)
elif (test):
    statement(s)
eelif (test):
    statement(s)
```

- Example:

```python
if (place == 1):
    print("Gold medal!")
eelif (place == 2):
    print("Silver medal!")
eelif (place == 3):
    print("Bronze medal.")
```
Nested if structures

- **exactly 1 path** *(mutually exclusive)*
  
  ```python
  if (test):
      statement(s)
  elif (test):
      statement(s)
  else:
      statement(s)
  ```

- **0 or 1 path** *(mutually exclusive)*
  
  ```python
  if (test):
      statement(s)
  elif (test):
      statement(s)
  else:
      statement(s)
  ```

- **0, 1, or many paths** *(independent tests; not exclusive)*
  
  ```python
  if (test):
      statement(s)
  if (test):
      statement(s)
  if (test):
      statement(s)
  if (test):
      statement(s)
  ```
Which nested if/else?

• (1) if/if/if  (2) nested if/else  (3) nested if/else/if
  
  • Whether a user is lower, middle, or upper-class based on income.
    • (2) nested if / else if / else

  • Whether you made the dean’s list (GPA ≥ 3.8) or honor roll (3.5-3.8).
    • (3) nested if / else if

  • Whether a number is divisible by 2, 3, and/or 5.
    • (1) sequential if / if / if

  • Computing a grade of A, B, C, D, or F based on a percentage.
    • (2) nested if / else if / else if / else if / else
Factoring if/else code

• **factoring**: Extracting common/repeated code.
  • Can reduce or eliminate repetition from if/else code.

• Example:

```python
if (a == 1):
    print(a)
    x = 3
    b = b + x
elif (a == 2):
    print(a)
    x = 6
    y = y + 10
    b = b + x
else:  # a == 3
    print(a)
    x = 9
    b = b + x
```

```python
print(a)
x = 3 * a
if (a == 2):
    y = y + 10
    b = b + x
```
The "dangling if" problem

• What can be improved about the following code?
  
  ```python
  if (x < 0):
      print("x is negative")
  elif (x >= 0):
      print("x is non-negative")
  ```

• The second if test is unnecessary and can be removed:

  ```python
  if (x < 0):
      print("x is negative")
  else:
      print("x is non-negative")
  ```

• This is also relevant in functions that use if with return...
if/else with return

```python
# Returns the larger of the two given integers.
def max(a, b):
    if (a > b):
        return a
    else:
        return b
```

• Functions can return different values using if/else
  • Whichever path the code enters, it will return that value.
  • Returning a value causes a function to immediately exit.
  • All paths through the code should reach a return statement.
Nested if/else question

• Write a program that produces output like the following:

This program reads data for two students and computes their Computer Science GPAs

Enter next person's information:
CS 110 grade? A
CS 120 grade? B

Enter next person's information:
CS 110 grade? B
CS 120 grade? B

Person 1 GPA = 3.5
accepted
Person 2 GPA = 3.0
accepted
Difference = 0.5

<table>
<thead>
<tr>
<th>Grade</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4.0</td>
</tr>
<tr>
<td>B</td>
<td>3.0</td>
</tr>
<tr>
<td>C</td>
<td>2.0</td>
</tr>
<tr>
<td>D</td>
<td>1.0</td>
</tr>
</tbody>
</table>