Conditionals

Overview: Making Decisions

If “is it raining”:
  take the umbrella
  wear rainboots
  wear raincoat
Else:
  wear sandals
  wear a summer dress

“Is it raining” is an expression that can return True or False.
In a Python program we can use:
- True/False values
- Relational Expressions
- Logical Expressions
- Predicates (all evaluate to True/False) whenever the code needs to make a decision for what to do next.

Flow Diagrams

The Road Not Taken
Two roads diverged in a yellow wood,  
And sorry I could not travel both

IMPORTANT: Only one of the branches is ever executed when a conditional statement is encountered. That is what the Flow Diagram exemplifies.

Robert Frost
Expressing the Same Function Two Ways

Are these two functions logically equivalent? Do they return the same answer for all inputs?

```python
def abs(n):
    '''returns absolute value'''
    if n < 0:
        return -n
    else:
        return n
```

```python
def abs(n):
    '''returns absolute value'''
    if n < 0:
        return -n
    return n
```

Notice the missing `else`.

A Better Approach: Chained Conditionals

```python
if boolean_expression1:
    statement1
    statement2
elif boolean_expression2:
    statement3
    statement4
elif boolean_expression3:
    statement5
    statement6
else:
    statement7
    statement8
```

```python
def movieAge(age):
    if age < 8:
        return 'G'
    elif age < 13:
        return 'PG'
    elif age < 18:
        return 'PG-13'
    else:
        return 'R'
```

Flow Diagram: Chained Conditionals

```
True
  ↓
test1 True
  ↓
statement1
```

```python
if boolean_expression1:
    statement1
    statement2
elif boolean_expression2:
    statement3
    statement4
elif boolean_expression3:
    statement5
    statement6
else:
    statement7
    statement8
```

```
True
  ↓
test2 True
  ↓
statement3
```

A Better Approach: Chained Conditionals

Compare this implementation of `movieAge` with that of the previous slide. For chained conditionals, we write less code, which is also easier to read because of fewer indentations.

**IMPORTANT:** In the moment one of the tests is True, the associated statements are executed and the chained conditional is exited. Only in the case when tests are False, we continue checking to find a True test.
Exercises [For notebook in class]

**Exercise 1:** Define a function named `letterGrade` that takes one score (the average of all your individual scores in a class), and returns a letter grade.

Assume:
A >= 90, B >= 80, C >= 70, D >= 60, F < 60

**Exercise 2:** Define a function named `addArticle` that takes a string argument and returns a new string with the correct article (a or an) added to the front of the argument.

**Exercise 3:** Define a function named `daysInMonth` that takes a month (as an integer between 1, for January, and 12, for December) as the parameter, and returns the number of days in it, assuming the year is not a leap year. If the month does not fall between 1 and 12, return an error message as a string. Make the function as concise as possible

---

isVowel revisited

The following definition doesn’t work. Why?

```python
def isVowel(s):
    l = s.lower()
    return l == ('a' or 'e' or 'i' or 'o' or 'u')
```

Because by Python’s treatment of truthy/falsey values, it’s equivalent to

```python
def isVowel(s):
    l = s.lower()
    return l == 'a'
```

---

All Python values are either Truthy or Falsey

Unexpectedly, in the context of if, and, or, Python treats a small number of so-called Falsey values (0, '', None, [], (), and {}) as False and all other values as True (so-called Truthy values).

In general, we think it is bad style to write code that depends on this fact; use Boolean expressions instead!

```python
def testTruthy(val):
    if val:
        return 'Truthy'
    else:
        return 'Falsey'
testTruthy(True)  # 'Truthy'
testTruthy(False) # 'Falsey'
testTruthy(17)    # 'Truthy'
testTruthy(0)    # 'Falsey'
testTruthy('hello')  # 'Truthy'
testTruthy('')     # 'Falsey'
testTruthy(None)  # 'Falsey'
testTruthy([1,2,3]) # 'Truthy'
testTruthy([])   # 'Falsey'
```

---

Simplifying Boolean Expressions and Conditionals

There are several code patterns involving boolean expressions and conditionals that can be simplified. The unsimplified versions are considered to be bad style and will be flagged by our Codder tool. Below BE stands for any expression evaluating to a boolean, and STMS stands for any statements.

<table>
<thead>
<tr>
<th>Complex Exp/Stmt</th>
<th>Simpler Exp/Stmt</th>
<th>Complex Exp/Stmt</th>
<th>Simpler Exp/Stmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>BE == True</td>
<td>BE</td>
<td>BE == False</td>
<td>not BE</td>
</tr>
<tr>
<td>if BE:</td>
<td>return True</td>
<td>if BE:</td>
<td>return False</td>
</tr>
<tr>
<td></td>
<td>else:</td>
<td></td>
<td>return True</td>
</tr>
<tr>
<td>if BE1:</td>
<td>return BE1</td>
<td>if BE1:</td>
<td>return True</td>
</tr>
<tr>
<td></td>
<td>else:</td>
<td></td>
<td>return BE2</td>
</tr>
<tr>
<td>if BE; STMS</td>
<td>return True</td>
<td>result = BE</td>
<td>return BE</td>
</tr>
<tr>
<td></td>
<td>else:</td>
<td></td>
<td>return result</td>
</tr>
<tr>
<td></td>
<td>STMS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>return False</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Digging Deeper

Conditionals 7-9

Digging Deeper

Conditionals 7-10

Digging Deeper

Conditionals 7-11

Digging Deeper

Conditionals 7-12
Simplifying Boolean Expressions and Conditionals: Example

```python
def doesNotBeginWithVowel(s):
    if isVowel(s[0]) == False
        return True
    else:
        return False
```

```python
def doesNotBeginWithVowel(s):
    if not isVowel(s[0])
        return True
    else:
        return False
```

```python
def doesNotBeginWithVowel(s):
    return not isVowel(s[0])
```