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.

Conditionals (if Statements)

Boolean expressions are used to choose between two courses of action in a conditional statement introduced by the keyword if.

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

Above is the Python syntax for expression conditional statements. To notice:
- Colons at the end of line for if and else
- Indentation for lines succeeding if and else

Flow Diagrams

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

Robert Frost

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

When writing functions where both branches contain a `return` statement, we can drop the keyword `else`, as shown below. This is because `return` with exit the function (when `if` is True), or execute the rest, in case it's False. Exactly what `else` would do.

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

Notice the missing `else`

A Better Approach: Chained Conditionals

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

```
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

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

**Concepts in this slide:**
- **Nested Conditionals**: Syntax for nested conditionals, example of nesting.
- **A Better Approach**: Chained Conditionals
  - New keyword: `elif`. Replace nesting with chaining of conditionals.
- **Flow Diagram**: Another example of the flow diagram model for branched execution.

**IMPORTANT**: In the moment one of the tests is `True`, the associated statements are executed and the chained conditional is exited. We never execute statements associated with tests that evaluate to `False`.

**Conditionals**: 7-5, 7-6, 7-7, 7-8
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) 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

---

All Python values are either Truthy or Falsey

Unexpectedly, in the context of `if`, `and`, 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!

```
def testTruthy(val):
    if val:
        return 'Truthy'
    else:
        return 'Falsey'
testTruthy(True)  # 'Truthy'
testTruthy(False) # 'Falsey'
testTruthy(17)     # 'Truthy'
testTruthy(0)     # 'Falsey'
testTruthy('')    # 'Falsey'
testTruthy([])    # 'Falsey'
testTruthy([1,2,3]) # 'Truthy'
testTruthy([])    # 'Falsey'
```

Unexpectedly, in the context of `if`, `and`, 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!

```
Complex Expr/Stmt  Simpler Expr/Stmt  Complex Expr/Stmt  Simpler Expr/Stmt
BE == True         BE               BE == False       not BE
if BE:             return BE         if BE:             return False  return not BE
    return True else: return False
if BE1:            return BE1 and BE2 if BE1:            return True else: return BE2
    return BE2 else: return False
if BE:             STMS              if BE:             result = BE
    return True else: STMS return False
    return result
    return BE
```

---

**isVowel revisited**

The following definition doesn't work. Why?

```
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

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

---

Simulating 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.

```
Complex Expr/Stmt  Simpler Expr/Stmt  Complex Expr/Stmt  Simpler Expr/Stmt
BE == True         BE               BE == False       not BE
if BE:             return BE         if BE:             return False  return not BE
    return True else: return False
if BE1:            return BE1 and BE2 if BE1:            return True else: return BE2
    return BE2 else: return False
if BE:             STMS              if BE:             result = BE
    return True else: STMS return False
    return result
    return BE
```
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])
```