CS 17700
Boolean and Control Statements
Week 4
Announcements

- Read Chapter 4 (functions)
- Read Chapter 5 (conditionals)
- Work on Project 1
ANY QUESTIONS?
Table of Contents

- Review (print vs return)
- Booleans
- Decision structures
- Logical operators
- Argument Typing
**print vs return**

- These two functions can be very confusing.

```python
>>> def myFunc():
    print("this is a test")
    return "Another test"

>>> myFunc()
this is a test
'Another test'
```


**print vs return**

- **print** takes the given inputs and displays their values to the screen. In this example the input string ‘this is a test’ is output to screen.

- **return** passes the given value(s) back to the caller.
  - If you call a function from IDLE, then IDLE is the caller.
  - IDLE will display the return values from functions that are not assigned to variables.
  - This is done by IDLE to make debugging easier, otherwise you would have to add another `print` statement to show what the function returned.
**print vs return**

```python
>>> def myFunc():
    print("this is a test")
    return "Another test"

>>> myFunc()
this is a test
'Another test'
>>> a = myFunc()
this is a test
```

- **Output from print**: `this is a test'
- **Output from return**: is displayed by IDLE on screen since it is not assigned to variable
- If function call is assigned to variable the ‘returned’ value is assigned to ‘a’ and not displayed on screen by IDLE
>>> def myFunc():
    print("this is a test")
    return "Another test"

>>> myFunc()
this is a test
'Another test'

>>> a = myFunc()
this is a test

>>> def myFunc2():
    myFunc()

>>> myFunc2()
this is a test

• Since `myFunc2()` calls `myFunc()` it is the caller.

• The value returned by `myFunc()` is given to `myFunc2()` . Not to IDLE

• For this reason “Another test” is not displayed to screen
Booleans

- Boolean (logical) expressions:
  - An expression that can be evaluated as True or False

- We use logical expression in everyday language:
  - *If it is sunny today, then I should not need an umbrella.*
  - “it is sunny today” is a logical expression: its value can be either True or False.

- Code-like examples:
  - assume x=4
    - x>3
    - (True)
  - assume str=“abc”
    - type(str)==int
    - (False)
Boolean (AND)

\[ a \text{ and } b \]
\[ a \text{ and } \text{true} \]
\[ x > 0 \text{ and } x \leq 2 \]
\[ y > 0 \text{ and } y \geq 3 \text{(overlapped)} \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Q</td>
<td>(P \text{ and } Q)</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---------------------</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Booleans (OR)

a or b
a or true
x \leq 0 or x > 2
x > 5 or x < 10 (always true)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>P or Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Boolean (NOT)

not a
not (not a)
not x > 3
DeMorgan’s law
not (a or b) == (not a) and (not b)
not (a and b) == (not a) or (not b)

<table>
<thead>
<tr>
<th>P</th>
<th>not P</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Booleans (XOR)

- \((P \text{ and } \neg Q) \text{ or } ((\neg P) \text{ and } Q)\)
- It has a name: XOR (exclusive or)
- Can you do this?

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P</strong></td>
<td><strong>Q</strong></td>
<td><strong>P xor Q</strong></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>
Booleans (XOR)

- \((P \text{ and } \neg Q) \text{ or } ((\neg P) \text{ and } Q)\)
- It has a name: XOR (exclusive or)
- Can you do this?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>( P \text{ xor } Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Decision Structures

- Now that we understand Booleans what good are they?
  - They can be used in decision structures!!!!!!
If statement

- An if statement takes a logical expression and evaluates it.
- If it is True, the statements in the if block are executed.
- If it is False, they are not executed.

Simple decision examples

```
x = 5
if x>1:
    print("print something")

The string is printed
```

```
x = 0
if x>1:
    print("print something")

Does nothing!
```
Decision Structures

- Two-way decisions are designated by if, else blocks
- If the expression is True the if block is executed, otherwise the else block is executed

```
a = 45
if a < 100:
    print("a is small")
else:
    print("a is large")

>>> a is small
```

```
a = 153
if a < 100:
    print("a is small")
else:
    print("a is large")

>>> a is large
```
Two-way decision

Decision Structures

"a is large"

"a is small"

a < 100?
Decision Structures

- Multi-way decision
- Decision statements can be nested within one another creating complex logic

```python
a = 1.5
if a > 2:
    print("a>2")
else:
    if a > 1:
        print("1<a<=2")
    else:
        print("a<=1")

>>> 1<a<=2
```
Decision Structures

- Multi-way decision

```
a >2 ?

no

a >1 ?

no

"a<1"

yes

"1<a<=2"

yes

"a>2"
```
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Checks if the two values are equal or not, if yes then condition becomes true.</td>
<td>a == b</td>
</tr>
<tr>
<td>!=</td>
<td>Checks if the two values are equal or not, if values are not equal condition becomes true</td>
<td>a != b</td>
</tr>
<tr>
<td>&lt;</td>
<td>Checks if the left value is less than the right value. If yes then condition is true</td>
<td>a &lt; b</td>
</tr>
<tr>
<td>&gt;</td>
<td>Checks if the left value is greater than the right value. If yes then condition is true</td>
<td>a &gt; b</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Checks if the left value is less than OR equal to right value. If yes then condition is true.</td>
<td>a &lt;= b</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Checks if the left value is greater than OR equal to the right value. If yes then condition is true.</td>
<td>a &gt;= b</td>
</tr>
</tbody>
</table>
Argument Typing

- Functions typically assume something important about the arguments
  
  ```python
  def displayInfo(a, b, c):
      print(“My Name is:”, a)
      print(“And I have”, b+c, “Children”)  
  ```

- Will this work no matter what we provide as arguments?

- Consider:
  
  ```python
  displayInfo(1, 2, 3)
  displayInfo(“Mary”, 4, 5)
  displayInfo(“John”, 1, “Test”)  
  ```
Argument Typing

There are two ways to handle this difficulty

1. Define what the function expects
2. Have the function check the arguments to ensure they are as expected

A combination of both techniques should be used
Function Documentation

- This solution uses comments to define what it expects
- This is good programming practice so other users know what to expect when calling your function

```python
# This function expects three inputs
# a -> Name of person which is a string
# b -> Number of male children which is an integer
# c -> Number of female children which is an integer
def displayInfo(a,b,c):
    print("My Name is:",a)
    print("And I have",b+c,"Children")
```
This solution uses comments and if/else statements to ensure each argument is of the expected type.

```python
# This function expects three inputs
# a -> Name of person which is a string
# b -> Number of male children which is an integer
# c -> Number of female children which is an integer
def displayInfo(a, b, c):
    if type(a) == str and type(b) == int and type(c) == int:
        print("My Name is:", a)
        print("And I have", b+c, "Children")
    else:
        print("Type error")
```
ANY QUESTIONS?