CS 17700
Boolean Statements and Decision Structures

Week 6
Announcements

- Midterm 1 is on OCT 9 6:30pm - 7:30pm CL50 224
  - Let us know in advance about conflicts or other valid makeup excuses.
  - Make sure to bring your Purdue ID and a pencil
Table of Contents

- Review (print vs. return)
- Online Python Tutor
- Logical operators
- Booleans
- Decision structures
- 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` is displayed by IDLE on screen since it is not assigned to variable.

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

• The value returned by `myFunc()` is given to `myFunc2()`.

• For this reason “Another test” is not displayed to screen.
Returning multiple values

def func(x):
    y_mul=x*2
    y_div=x/2
    y_pow=x**2
    return y_mul, y_div, y_pow

a, b, c=func(4)
a=?, b=?, c=?
```python
>>> def getabc():
    a="Hello"
    b="world"
    c="!"
    return a, b, c

>>> getabc()
('Hello', 'world', '!!')
>>> def func(x):
    y_mul=x*2
    y_div=x/2
    y_pow=x**2
    return y_mul, y_div, y_pow

>>> a, b, c=func(4)
>>> print (a)
8
>>> print (b)
2.0
>>> print (c)
16
>>> func(4)
(8, 2.0, 16)
>>> a=func(4)
>>> print (a)
(8, 2.0, 16)
>>> a, b=func(4)
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    a, b=func(4)
ValueError: too many values to unpack (expected 2)
>>> a, b, c, d=func(4)
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    a, b, c, d=func(4)
ValueError: need more than 3 values to unpack
```
Online Python Tutor

- Using this tool, a teacher or student can write a Python program in the Web browser and visualize what the computer is doing step-by-step as it executes the program.

- You can use this tool from this website:
  - [http://pythontutor.com](http://pythontutor.com)
Online Python Tutor Example

- Two functions are defined to print some statements.
- Name of functions: sing and happy
- sing is called from main function
- Happy functions is called from sing function.
Online Python Tutor Example

```python
# From "Teaching with Python" by John Zelle

def happy():
    print("Happy Birthday to you!")

def sing(P):
    happy()
    happy()
    print("Happy Birthday dear " + P + "!")
    happy()

# main
sing("Fred")
```

- Steps which are executed
- Sing function called from main
Online Python Tutor Example (cont)

```python
# From "Teaching with Python" by John Zelle

def happy():
    print("Happy Birthday to you!")

def sing(p):
    happy()
    happy()
    print("Happy Birthday dear " + p + "!")
    happy()

# main
sing("Fred")
```

- Happy function is called
Online Python Tutor Example (cont)

Happy function is executed one time and prints “Happy Birthday”
Online Python Tutor Example (cont)

```python
# From "Teaching with Python" by John Zelle

def happy():
    print("Happy Birthday to you!")

def sing(P):
    happy()
    happy()
    print("Happy Birthday dear " + P + "!")
    happy()

# main
sing("Fred")
```

All of the steps are executed and program terminated.
## Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>Checks if the two values are equal or not, if yes then condition becomes true</td>
<td><code>a == b</code></td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Checks if the two values are equal or not, if values are not equal condition becomes true</td>
<td><code>a != b</code></td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Checks if the left value is less than the right value. If yes then condition is true</td>
<td><code>a &lt; b</code></td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Checks if the left value is greater than the right value. If yes then condition is true</td>
<td><code>a &gt; b</code></td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Checks if the left value is less than OR equal to right value. If yes then condition is true</td>
<td><code>a &lt;= b</code></td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Checks if the left value is greater than OR equal to the right value. If yes then condition is true</td>
<td><code>a &gt;= b</code></td>
</tr>
</tbody>
</table>
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.*
  • Here *is it sunny today?* is a logical expression: its value can be either **True** or **False**

• Code-like examples:
  
  Assume `x=4`  
  `x>3`  
  Result-**True**

  Assume `a_string=“abc”`  
  `type(a_string)==int`  
  Result-**False**
Boolean (AND)

Examples:
Suppose \(a=\text{True}, \ b=\text{False}\):
- \(a \text{ and } b=?\)
- \(a \text{ and } \text{True}=?\)

Suppose \(x=1, \ y=1\):
- \(x > 0 \text{ and } x \leq 2\) – Result?
- \(y > 0 \text{ and } y \geq 3\) – Result?

<table>
<thead>
<tr>
<th>(x)</th>
<th>(y)</th>
<th>(x \text{ and } y)</th>
</tr>
</thead>
<tbody>
<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)

Examples:-
Suppose $a = \text{True}$, $b = \text{False}$
a or b=?
a or True=?

Suppose $x = 1$
x <= 0 or x > 2
Result?
x > 5 or x < 10
Result?

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$x \ or \ y$</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>
Examples:-
Suppose \(a=\text{True}, x=2\)

- \(\text{not } a=?\)
- \(\text{not } (\text{not } a)=?\)
- \(\text{not } x > 3=?\)

DeMorgan’s law:-

- \(\text{not } (a \text{ or } b) = (\text{not } a) \text{ and } (\text{not } b)\)
- \(\text{not } (a \text{ and } b) = (\text{not } a) \text{ or } (\text{not } b)\)

<table>
<thead>
<tr>
<th></th>
<th>(\text{not } x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Decision Structures

Why do we need Booleans?

- They can be used in decision structures

- When used in the decision structure, the subsequent code will be executed only when the Boolean expression which is evaluated as the condition turns out to be ‘True’
### 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

```python
x = 5
if x > 1:
    print("print something")
# The string is printed

x = 0
if x > 1:
    print("print something")
    print("Does it print?")
# 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

```python
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
```
Decision Structures

- Two-way decision

\[ a < 100? \]

- no → "a is small"
- yes → "a is large"
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”
```
What will be printed?

credits=78
GPA=3.5

if credits >= 120 and GPA >= 2.0:
    print('You are eligible to graduate!')
else:
    print('You are not eligible to graduate.')
What will be printed?

\[
a = \textit{*made up integer value*}, \quad b = \textit{*made up integer value*}
\]

if \(a \geq 0\) and \(b \geq 0\):

    print ("Both a and b are positive")

elif \(a < 0\) and \(b \geq 0\):

    print ("a is negative, b is positive")

elif \(a < 0\) and \(b < 0\):

    print ("Both a and b are negative")

else:

    print ("a is positive, b is negative")
Print Even Numbers

How we can print even numbers using while and if statement?
Print Even Numbers

```python
def main():
    print("We will show the even numbers up to 20")
    n = 1
    while n <= 20:
        if n % 2 == 0:
            print(n)
        n = n + 1
    print("there, done.")

main()
```

This condition checks the number is even or not
Print Even Numbers (output)

```python
Python 3.2 (r32:88445, Feb 20 2011, 21:30:00) [MSC v.1500 64 bit (AMD64)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ___________________________________________________________
>>> __________________________________________________________
>>> We will show the even numbers up to 20
2
4
6
8
10
12
14
16
18
20
there, done.
>>> |
```
One more Example

```python
a = 10
while a > 0:
    print(a)
    if a > 5:
        print("Big number!")
    elif a % 2 != 0:
        print("This is an odd number")
        print("It isn't greater than five, either")
    else:
        print("this number isn't greater than 5")
        print("nor is it odd")
        print("feeling special?")
    a = a - 1
    print("we just made 'a' one less than what it was!")
    print("and unless a is not greater than 0, we'll do the loop again.")
print("well, it seems as if 'a' is now no bigger than 0!")
print("the loop is now over, and without further adieu, so is this program!")
```
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")
```
Argument Typing Checks

- 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?