Lecture 02
Making Decisions: Conditional Execution
Flow of Control

• Flow of control = order in which statements are executed

• By default, a program's statements are executed sequentially, from top to bottom.

```
program
total = 0
num1  = 5
num2  = 10
total = num1 + num2
```

```
flowchart

total = 0

num1 = 5

num2 = 10

total = num1 + num2
```
Conditional Execution

• To solve many types of problems we need to change the standard flow of control

• Conditional execution allows you to decide whether to do something, based on some condition
  • example:

```python
def abs_value(x):
    """ returns the absolute value of input x """
    if x < 0:
        x = -1 * x
    return x
```

• examples of calling this function from the Shell:

```python
>>> abs_value(-5)
5
>>> abs_value(10)
10
```
Simple Decisions: if Statements

- Syntax:
  
  ```python
  if condition:
      true block
  ```

  where:

  - `condition` is an expression that is true or false
  - `true block` is one or more indented statements

- Example:

  ```python
  def abs_value(x):
      
      """ returns the absolute value of input x ""
      if x < 0:
          
          # true block
          x = -1 * x
      
      return x
  ```
Two-Way Decisions: if-else Statements

- Syntax:
  ```python
  if condition:
      true block
  else:
      false block
  ```

- Example:
  ```python
def pass_fail(avg):
    
    """ checks whether student passes/fails """
    if avg >= 60:
        grade = 'pass'    # true block
    else:
        grade = 'fail'    # false block
    return grade
  ```
A Word About Blocks

• A block can contain multiple statements.

```python
def welcome(class):
    if class == 'frosh':
        print('Welcome to Brown U!')
        print('Have a great four years!')
    else:
        print('Welcome back!')
        print('Have a great semester!')
        print('Be nice to the frosh students.')
```

• A new block begins whenever we increase the amount of indenting.

• A block ends when we either:
  • reach a line with less indenting than the start of the block
  • reach the end of the program
Expressing Simple Conditions

Python provides a set of *relational operators* for making comparisons:

<table>
<thead>
<tr>
<th>operator</th>
<th>name</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>val &lt; 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>price &lt; 10.99</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>num &gt; 60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>state &gt; 'Ohio'</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>average &lt;= 85.8</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>name &gt;= 'Jones'</td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
<td>total == 10</td>
</tr>
<tr>
<td>!==</td>
<td>not equal to</td>
<td>age != my_age</td>
</tr>
</tbody>
</table>

*don't confuse `==` with `='*
Boolean Expressions

• A condition has one of two values: True or False.
  
  >>> 10 < 20
  True
  >>> 10 < 20 < 15
  False
  >>> "Jones" == "Baker"
  False

• True and False are not strings.
  • they are literals from the bool data type
    >>> type(True)
    <class 'bool'>
    >>> type(30 > 6)
    <class 'bool'>

• An expression that evaluates to True or False is known as a boolean expression.
Forming More Complex Conditions

- Python provides *logical operators* for combining/modifying boolean expressions:

<table>
<thead>
<tr>
<th>name</th>
<th>example and meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>age &gt;= 18 and age &lt;= 35</td>
</tr>
<tr>
<td></td>
<td>True if both conditions are True; False otherwise</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>or</th>
<th>age &lt; 3 or age &gt; 65</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>True if one or both of the conditions are True; False if both conditions are False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>not</th>
<th>not (grade &gt; 80)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>True if the condition is False; False if it is True</td>
</tr>
</tbody>
</table>
Nesting

- We can "nest" one conditional statement in the true block or false block of another conditional statement.

```python
def welcome(class):
    if class == 'frosh':
        print('Welcome to BU!')
        print('Have a great four years!')
    else:
        print('Welcome back!')
        if class == 'senior':
            print('Have a great last year!')
        else:
            print('Have a great semester!')
    print('Be nice to the frosh students!')
```
What is the output of this program?

```python
x = 5
if x < 15:
    if x > 8:
        print('one')
    else:
        print('two')
else:
    if x > 2:
        print('three')
```

A. one
B. two
C. three
D. more than one of the above
E. nothing is output
What is the output of this program?

```
x = 5
if x < 15:    # true
    if x > 8:  # false
        print('one')
    else:
        print('two')
else:
    if x > 2:  # program would go here next...
        print('three')
```

A. one
B. two
C. three
D. more than one of the above
E. nothing is output
What does this print? (note the changes!)

```python
x = 5
if x < 15:
    if x > 8:
        print('one')
    else:
        print('two')
if x > 2:
    print('three')
```

A. one
B. two
C. three
D. more than one of the above
E. nothing is output
What does this print? (note the changes!)

```python
x = 5
if x < 15:
    if x > 8:
        print('one')
    else:
        print('two')
if x > 2:
    print('three')
```

A. one  
B. two  
C. three  
D. more than one of the above  
E. nothing is output
What does this print? (note the new changes!)

```python
x = 5
if x < 15:
    if x > 8:
        print('one')
    else:
        print('two')
if x > 2:
    print('three')
```

A. one
B. two
C. three
D. more than one of the above
E. nothing is output
What does this print? (note the new changes!)

```python
x = 5
if x < 15:
    if x > 8:
        print('one')
    else:
        print('two')
else:
    print('three')
if x > 2:
    print('three')
```

A. one  
B. two  
C. three  
D. more than one of the above  
E. nothing is output
• The following function doesn't work.

```python
def letter_grade(avg):
    if avg >= 90:
        grade = 'A'
    if avg >= 80:
        grade = 'B'
    if avg >= 70:
        grade = 'C'
    if avg >= 60:
        grade = 'D'
    else:
        grade = 'F'
    return grade
```

• example:

```python
>>> letter_grade(95)
'D'
```
Multi-Way Decisions (cont.)

• Here's a fixed version:

```python
def letter_grade(avg):
    if avg >= 90:
        grade = 'A'
    elif avg >= 80:
        grade = 'B'
    elif avg >= 70:
        grade = 'C'
    elif avg >= 60:
        grade = 'D'
    else:
        grade = 'F'
    return grade
```

• example:
```python
>>> letter_grade(95)
'A'
```
Multi-Way Decisions: if-elif-else Statements

• Syntax:

```python
if condition1:
    true block for condition1
elif condition2:
    true block for condition2
elif condition3:
    true block for condition3
...
else:
    false block
```

• The conditions are evaluated in order. The true block of the first true condition is executed.

• If none of the conditions are true, the false block is executed.
Flowchart for an if-elif-else Statement

- **condition 1**
  - true: true block 1
  - false: false block
- **condition 2**
  - true: true block 2
  - false: ... (ellipses indicating continuation of false block)
- **false block**
- Next statement
How many lines does this print?

```python
x = 5
if x == 8:
    print('how')
elif x > 1:
    print('now')
elif x < 20:
    print('brown')
print('cow')
```

A. 0
B. 1
C. 2
D. 3
E. 4
How many lines does this print?

```python
x = 5
if x == 8:
    print('how')
elif x > 1:
    print('now')
elif x < 20:
    print('brown')
print('cow')
```

A. 0
B. 1
C. 2
D. 3
E. 4
How many lines does this print?

```python
x = 5
if x == 8:
    print('how')
if x > 1:
    print('now')
if x < 20:
    print('brown')
print('cow')
```

A. 0  
B. 1  
C. 2  
D. 3  
E. 4  

23
How many lines does this print?

```python
x = 5
if x == 8:
    print('how')
if x > 1:
    print('now')
if x < 20:
    print('wow')
print('cow')
```

A. 0
B. 1
C. 2
D. 3
E. 4
What is the output of this code?

def mystery(a, b):
    if a == 0 or a == 1:
        return b
    return a * b

print(mystery(0, 5))

A. 5
B. 1
C. 0
D. none of these, because an error is produced
E. none of these, but an error is not produced
What is the output of this code?

```
def mystery(a, b):
    if a == 0 or a == 1:
        return b
    return a * b

print(mystery(0, 5))
```

A. 5
B. 1
C. 0
D. none of these, because an error is produced
E. none of these, but an error is not produced
What is the output of this code?

```
def mystery(a, b):
    if a == 0 or a == 1:
        return b  # return 5
    return a * b

print(mystery(0, 5))  # print(5)
```

A. 5
B. 1
C. 0
D. none of these, because an error is produced
E. none of these, but an error is not produced

A return statement ends a function call, regardless of whether the function has more lines after the return.
def mystery(a, b):
    if a == 0 or 1:
        # this is problematic
        return b
    return a * b
print(mystery(0, 5))

• When using `and / or`, both sides of the operator should be a boolean expression that could stand on its own.

  
  boolean    boolean    boolean    integer
  a == 0 or  a == 1     a == 0 or 1
  (do this)             (don't do this)

• Unfortunately, Python doesn't complain about code like the problematic code above.
  • but it won't typically work the way you want it to!
Avoid Overly Complicated Code

• The following also involves decisions based on a person's age:

```python
age = ... # let the user enter his/her age
if age < 13:
    print('You are a child.')
elif age >= 13 and age < 20:
    print('You are a teenager.')
elif age >= 20 and age < 30:
    print('You are in your twenties.')
elif age >= 30 and age < 40:
    print('You are in your thirties.')
else:
    print('You are a survivor.')
```

• How could it be simplified?
Avoid Overly Complicated Code

• The following also involves decisions based on a person's age:

```python
age = ...  # let the user enter his/her age
if age < 13:
    print('You are a child.')
elif age >= 13 and age < 20:
    print('You are a teenager.')
elif age >= 20 and age < 30:
    print('You are in your twenties.')
elif age >= 30 and age < 40:
    print('You are in your thirties.')
else:
    print('You are a survivor.')
```

• How could it be simplified?
Variable Scope
Functions Calling Functions
Variable Scope

• The *scope* of a variable is the portion of your program in which the variable can be used.

• *We need to distinguish between:*
  • *local* variables: limited to a particular function
  • *global* variables: can be accessed anywhere
def mystery(x, y):
    b = x - y       # b is a local var of mystery
    return 2*b      # we can access b here

c = 7
mystery(5, 2)
print(b + c)    # we can't access b here!

• When we assign a value to a variable inside a function, we create a local variable.
  • it "belongs" to that function
  • it can't be accessed outside of that function

• The parameters of a function are also limited to that function.
  • example: the parameters x and y above
Global Variables

def mystery(x, y):
    b = x - y
    return 2*b + c  # works, but not recommended

c = 7            # c is a global variable
mystery(5, 2)    # we can access c here
print(b + c)     # we can access c here

• When we assign a value to a variable outside of a function, we create a global variable.
  • it belongs to the global scope

• A global variable can be used anywhere in your program.
  • in code that is outside of any function
  • in code inside a function (but this is not recommended)

Neither globals nor locals exist until they are assigned a value!
The program above has two different variables called b.

- one local variable
- one global variable

When this happens, the local variable has priority inside the function to which it belongs.
What is the output of this code?

def mystery2(a, b):
    x = a + b
    return x + 1

x = 8
mystery2(3, 2)
print(x)

A. 5
B. 6
C. 8
D. 9
E. none of these, because an error is produced
What is the output of this code?

def mystery2(a, b):  # there are two different x's!
    x = a + b        # this x is local to mystery2
    return x + 1

x = 8              # this x is global
mystery2(3, 2)
print(x)

A. 5
B. 6
C. 8
D. 9
E. none of these, because an error is produced
What is the output of this code?

def mystery2(a, b):  # there are two different x's!
    x = a + b  # this x is local to mystery2
    return x + 1

x = 8  # this x is global
mystery2(3, 2)
print(x)

Follow-up question:
Why don't we see the following?

A. 5
B. 6
C. 8
D. 9
E. none of these, because an error is produced
What is the output of this code?

def mystery2(a, b):  # there are two different x's!
    x = a + b         # this x is local to mystery2
    return x + 1

x = 8                # this x is global
mystery2(3, 2)
print(x)

Follow-up question:
Why don't we see the following?
6
8
mystery2(3, 2) returns 6, but we don't print the return value. We essentially "throw it away"!

A. 5
B. 6
C. 8
D. 9
E. none of these, because an error is produced
What is the output of this code? (version 2)

def mystery2(a, b):
    x = a + b
    return x + 1

x = 8
mystery2(3, 2)
print(x)

A. 5
B. 6
C. 8
D. 9
E. none of these, because an error is produced
What is the output of this code? *(version 2)*

def mystery2(a, b):
    x = a + b
    return x + 1

x = 8
mystery2(3, 2)
print(x)  # the only x belongs to mystery2,
          # so we can't access it here.

A.  5
B.  6
C.  8
D.  9
E. none of these, because an error is produced
A Note About Globals

• It's not a good idea to access a global variable inside a function.
  • for example, you shouldn't do this:
    ```python
def average3(a, b):
    total = a + b + c  # accessing a global c
    return total/3
```
```python
c = 7
print(average3(5, 7))
```

42
A Note About Globals

• It's not a good idea to access a global variable inside a function.
  • for example, you shouldn't do this:
    
    ```python
    def average3(a, b):
        total = a + b + c  # accessing a global c
    return total/3
    
    c = 7
    print(average3(5, 7))
    ```

• Instead, you should pass it in as a parameter/input:

    ```python
    def average3(a, b, c):
        total = a + b + c  # accessing input c
    return total/3
    
    c = 7
    print(average3(5, 7, c))
    ```
Frames and the Stack

- Variables are stored in blocks of memory known as \textit{frames}.

- Each function call gets a frame for its local variables.
  - goes away when the function returns

- Global variables are stored in the global frame.

- The \textit{stack} is the region of the computer's memory in which the frames are stored.
  - thus, they are also known as \textit{stack frames}
Visualizing How Functions Work
pythontutor.com/visualize.html

• Before the call to mystery2:

```python
1 def mystery2(a, b):
2     x = a + b
3     return x + 1
4
5 x = 8
6 mystery2(3, 2)
7 print(x)
```

The global frame includes the function names and the global variables.
• At the start of the call to `mystery2`:

```python
def mystery2(a, b):
    x = a + b
    return x + 1
```

`mystery2(3, 2)` gets its own frame containing the variables that belong to it. `mystery2's` `x` isn't shown yet because we haven't assigned anything to it.
Visualizing How Functions Work
pythontutor.com/visualize.html

• When the call to mystery2 is about to return:

```python
1 def mystery2(a, b):
2     x = a + b
3     return x + 1
4
5     x = 8
6     mystery2(3, 2)
7     print(x)
```

Python looks for a variable in the current frame first, so the local x will be used instead of the global x when returning x + 1.
Visualizing How Functions Work
pythontutor.com/visualize.html

• After the call to `mystery2` has returned:

```python
1  def mystery2(a, b):
2      x = a + b
3      return x + 1

5  x = 8

6  mystery2(3, 2)

7  print(x)
```

When a function call returns, its frame is removed from memory. Its local variables can no longer be accessed.

• The only `x` that remains is the global `x`, so its value is printed.
What is the output of this code?

def quadruple(y):
    y = 4 * y
    return y

y = 8
quadruple(y)

print(y)

A. 4
B. 8
C. 12
D. 32
E. none of these, because an error is produced
What is the output of this code?

def quadruple(y):    # the parameter y is local
    y = 4 * y
    return y

y = 8                # this y is global
quadruple(y)

print(y)

A. 4
B. 8
C. 12
D. 32
E. none of these, because an error is produced
What is the output of this code?

def quadruple(y):
    # 3. local y = 8
    y = 4 * y
    return y

y = 8
    # 1. global y = 8
quadruple(y)  # 2. pass in global y's value

print(y)

A. 4
B. 8
C. 12
D. 32
E. none of these, because an error is produced
What is the output of this code?

```python
def quadruple(y):
    # 3. local y = 8
    y = 4 * y
    # 4. local y = 4 * 8 = 32
    return y
    # 5. return local y's value

y = 8
quadruple(y) # 1. global y = 8
# 2. pass in global y's value
# 6. return value thrown away!

print(y)
```

A. 4  
B. 8  
C. 12  
D. 32  
E. none of these, because an error is produced
What is the output of this code?

def quadruple(y):
    # 3. local y = 8
    y = 4 * y
    # 4. local y = 4 * 8 = 32
    return y
    # 5. return local y's value
    y = 8
    quadruple(y)
    # 2. pass in global y's value
    # 6. return value thrown away!
    print(y)
    # 7. print global y's value,
    #    which is unchanged!

A. 4
B. 8
C. 12
D. 32
E. none of these, because an error is produced

You can't change the value of a variable by passing it into a function!
How could we change this to see the return value of `quadruple`?

```python
def quadruple(y):
    y = 4 * y
    return y

y = 8
quadruple(y)
print(y)
```
def quadruple(y):
    y = 4 * y
    return y

y = 8
y = quadruple(y)  # assign return val to global y
print(y)
Seeing the return value (option 2)

def quadruple(y):
    y = 4 * y
    return y

y = 8
print(quadruple(y))  # print return val
# no need for print(y)
What is the output of this program?

def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1 * x

print(demo(-4))

A. 4
B. 42
C. 44
D. 46
E. none of these
def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1 * x

print(demo(-4))

defining the function demo

<table>
<thead>
<tr>
<th>demo</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>ret</td>
<td>x</td>
</tr>
<tr>
<td>-4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

stack frame

x = -4
return -4 + f(-4)
def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1 * x

print(demo(-4))

These are distinct memory locations both holding \( x \)'s.

<table>
<thead>
<tr>
<th>demo</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>ret</td>
<td>x</td>
</tr>
<tr>
<td>-4</td>
<td>-4</td>
<td></td>
</tr>
</tbody>
</table>
def f(x):
    return 11*gil(x) + g(x//2)

def demo(x):
    return x + f(x)

def g(x):
    return -1 * x

print(demo(-4))

def demo(x):
    return x + f(x)

def f(x):
    return 11*gil(x) + g(x//2)

def g(x):
    return -1 * x

print(demo(-4))

---

<table>
<thead>
<tr>
<th>demo</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>ret</td>
<td>x</td>
</tr>
<tr>
<td>-4</td>
<td></td>
<td>-4</td>
</tr>
</tbody>
</table>
def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1 * x

print(demo(-4))

def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1 * x

print(demo(-4))

<table>
<thead>
<tr>
<th>demo</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>ret</td>
<td>x</td>
</tr>
<tr>
<td>-4</td>
<td></td>
<td>-4</td>
</tr>
</tbody>
</table>
def demo(x):
    return x + f(x)

def f(x):
    return 11 * g(x) + g(x // 2)

def g(x):
    return -1 * x

demo(-4)

These are distinct memory locations both holding x's – and now they also have different values!!

t | ret
t-4 | -4
-2 | -2

62
Functions Calling Other Functions!

```python
def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1 * x

print(demo(-4))
```

<table>
<thead>
<tr>
<th>x</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td>-2</td>
<td>2</td>
</tr>
</tbody>
</table>

These are distinct memory locations both holding x's – *and now they also have different values!*
def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1 * x

print(demo(-4))

demo        f         g
x | ret    x | ret   x | ret
-4 |       -4 |  46  -4 |  4
-2 |  2
functions calling other functions!

def demo(x):
    return x + f(x)

def f(x):
    return 11 * g(x) + g(x // 2)

def g(x):
    return -1 * x

print(demo(-4))
Functions Calling Other Functions!

def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1 * x

print(demo(-4))  # print(42)
42
What is the output of this program?

def demo(x):
    return x + f(x)

def f(x):
    return 11*g(x) + g(x//2)

def g(x):
    return -1 * x

print(demo(-4))

A. 4
B. 42
C. 44
D. 46
E. none of these
def foo(x, y):
    y = y + 1
    x = x + y
    print(x, y)
    return x

x = 2
y = 0

y = foo(y, x)
print(x, y)

foo(x, x)
print(x, y)

print(foo(x, y))
print(x, y)
def foo(x, y):
    y = y + 1
    x = x + y
    print(x, y)
    return x

x = 2
y = 0

y = foo(y, x)
print(x, y)

foo(x, x)
print(x, y)

print(foo(x, y))
print(x, y)
Tracing Function Calls

```python
def foo(x, y):
    y = y + 1
    x = x + y
    print(x, y)
    return x

x = 2
y = 0

y = foo(y, x)
print(x, y)

foo(x, x)
print(x, y)

print(foo(x, y))
print(x, y)
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

```
global

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
```

```
output

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
```