CSC 280 Introduction to Computer Science: Programming with Python

Lecture 6 : Conditionals, Control Flow
September 17 , 2015

Prof. Bei Xiao

Fall, 2015

American University

Outline

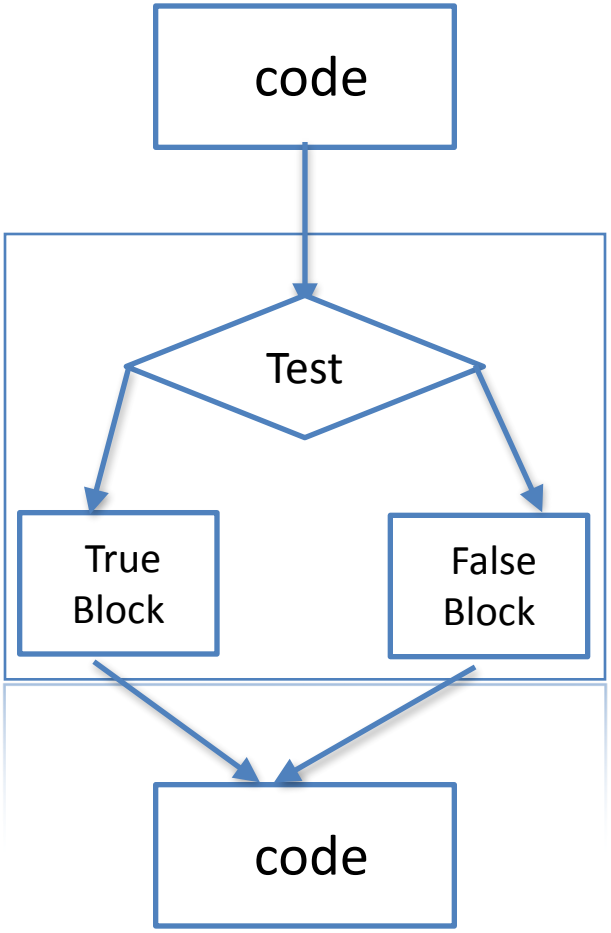
- Branching Programming
- Review of Boolean and Logical operator
- Conditionals and Control Flow

Branching Programs: Conditionals

- An ability to check conditions and change the behavior of the program accordingly, change ordering of instructions based on seem test.

```
if <some test>:
    block of instructions
else:
    Block of instructions
```

Branching Programs: Conditionals



Conditionals: If-Statement

Format

```
if <boolean-expression>:
    <statement>
    ...
    <statement>
```

Example

```
# Put x in z if it is positive
if x > 0:
    z = x
```

Boolean expression

Execution:

if *<booliean-expression>* is true, then execute all of the statements indented directly underneath (until first unindented statement)

Truth Table

expression	result
true `and` true	true
true `and` false	false
false `and` true	false
false `and` false	false

expression	result
`not` true	false
`not` false	true

expression	result
true `or` true	true
true `or` false	true
false `or` true	true
false `or` false	false

https://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_2.6/Boolean_Expressions

Comparisons

<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal (1)
is	object identity
is not	negated object identity

Boolean

- True and True
- False and True
- 1==1 and 2==1
- True and 1==1
- 1!=0
- “test” != “testing”
- not (True and False)
- not (1==1 and 0!=1)
- False or True
- 3==4 or 0!=1

Conditionals: If-Statement

Format

```
if <boolean-expression>:
    <statement>
    ...
    <statement>
```

Example

```
# Put x in z if it is positive
if x > 0:
    z = x
```

Boolean expression

Execution:

if *<booliean-expression>* is true, then execute all of the statements indented directly underneath (until first unindented statement)

Conditionals: If-else Statement

Format

```
if <boolean-expression>:
    <statement>
    ...
else:
    <statement>
    ...
```

Example

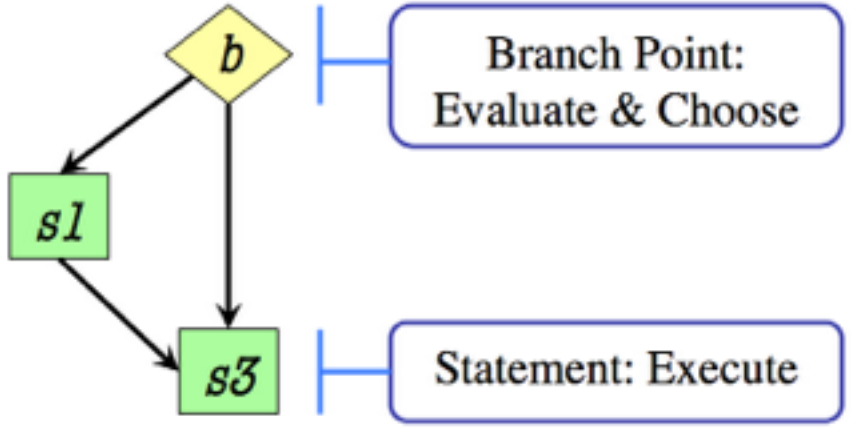
```
# Put max of x, y in z
if x > y:
    z = x
else:
    z = y
```

Execution:

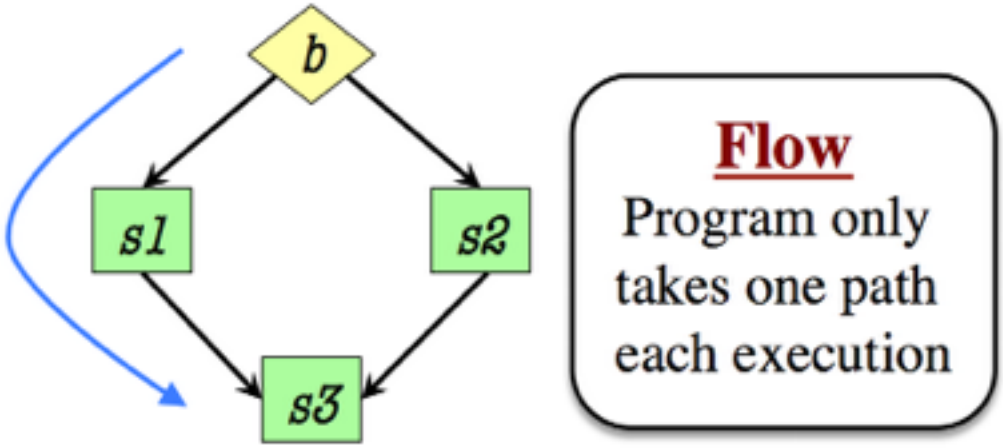
if *<booliean-expression>* is true, then execute statements indented under if; otherwise execute the statements indented under else

Conditionals: “Control Flow” Statements

```
if b:
    s1 # statement
s3
```



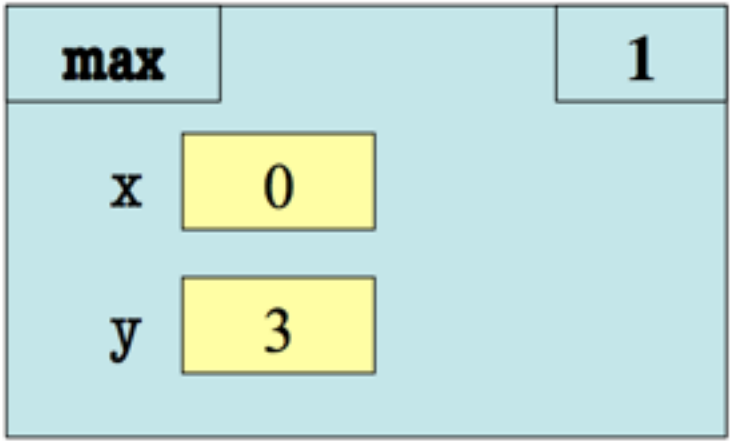

```
if b:
    s1
else:
    s2
s3
```



Program Flow and Call Frames

```
def max(x,y):
    """Returns: max of x, y"""
    # simple implementation
1   if x > y:
2       return x
3   return y
```

max(0,3):

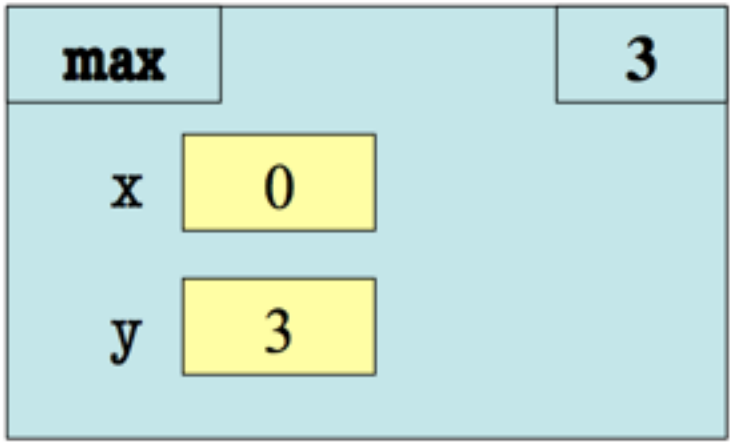


Frame sequence depends on flow

Program Flow and Call Frames

```
def max(x,y):
    """Returns: max of x, y"""
    # simple implementation
1   if x > y:
2       return x
3   return y
```

max(0,3):



Frame sequence depends on flow

Skips line 2

Variations of max(x,y)

```
def max(x,y):
    """Returns:
       max of x, y"""
    if x > y:
        return x
    else:
        return y
```

Which is better?
Matter of preference

There are two **returns**!
But only one is executed

Program Flow vs. Local Variables

```
def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
1   if x > y:
2       temp = x
3       x = y
4       y = temp


5   return y
```

Program Flow vs. Local Variables

```
def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
1   if x > y:
2       temp = x
3       x = y
4       y = temp


5   return y
```

Value of max (3,0)

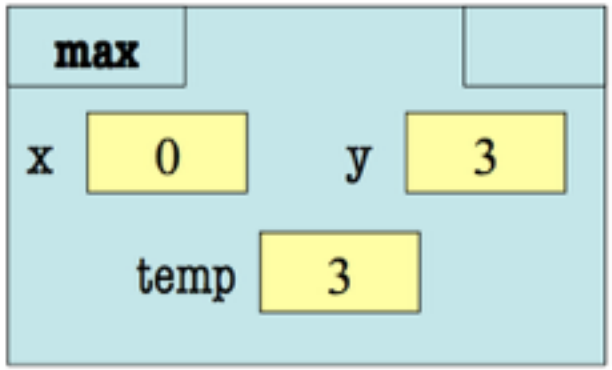
A. 3
B. 0
C. I don't know
D. Error!

Program Flow vs. Local Variables

```
def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
1   if x > y:
2       temp = x
3       x = y
4       y = temp

5   return y
```

- temp is needed for swap
  - x = y loses value of x
  - "Scratch computation"
  - Primary role of local vars
- max(3,0):



Program Flow vs. Local Variables

```
def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp

    return temp
```

Value of max (3,0)

A. 3
B. 0
C. I don't know
D. Error!

Program Flow vs. Local Variables

```
def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp

    return temp
```

Value of max (3,0)

A. 3 CORRECT
B. B. 0
C. I don't know
D. Error!

Program Flow vs. Local Variables

```
def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp

    return temp
```

Value of max (0,3)

A. 3
B. 0
C. I don't know
D. Error!

Program Flow vs. Local Variables

```
def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp

    return temp
```

Value of max (0,3)

A. 3
B. 0
C. I don't know
D. Error! CORRECT

- Variable existence depends on flow
- Understanding flow is important in testing

Program Flow vs. Local Variables

```
def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put larger in temp
    temp = y
    if x > y:
        temp = x

    return temp
```

First assigned

Inside scope

Value of max (0,3)

A. 3
B. 0
C. I don't know
D. Error!

Never refer to a variable that might not exit!!

Conditionals: If-Elif-Else-Statement

Format

```
if <boolean-expression>:
    <statement>
    ...
elif <boolean-expression>:
    <statement>
    ...
...
else:
    <statement>
    ...
```

Example

```
# Put max of x, y, z in w
if x > y and x > z:
    w = x
elif y > z:
    w = y
else:
    w = z
```

Must between if and else no limitations of numbers of **elif**

Conditionals: If-Elif-Else-Statement

Format

if <*boolean-expression*>:
    <*statement*>
    ...
elif <*boolean-expression*>:
    <*statement*>
    ...
...
else:
    <*statement*>
    ...

Example

- No limit on number of elif
  - Can have as many as want
  - Must be between if, else
- The else is always optional
  - if-elif by itself is fine
- Booleans checked in order
  - Once it finds a true one, it skips over all the others
  - else means **all** are false

Demo: find if a number is odd or even

```
def oddnumber(x):
    if (x%2) == 0:
        print "Even"
    else:
        print  "Odd"
        if x%3 != 0:
            print 'and not divisable by 3'

oddnumber(15)
```

Exercise

1. Prompt (using raw_input) to ask how many course credits a student have, if it has received more than 120 credits, print out that they can graduate!

2. Write an alternative code (a function) to check whether a number is even or odd. If it is even,

print “even”, and if it is odd, print “odd”.

Hint: Instead of using remainder, can you use “/“?

Exercise: Traffic light

Prompt user to ask about the color of the traffic light, if it is “red”, tell them to “stop”, if it is “yellow”, tell them “ slow down” if it is “green”, tell them “ go ahead”.

Hint: Use if ….elif…. else

Exercise

Ask the user to input their names, if it is your name, tell them that it is a nice name, if it is one of the stored name: Albert Einstein, George W Bush, Peter Pan, tell them What you feel about them, if it is not one of the stored names, tell them they have a nice name.

Hint: use raw_input String comparisons

if name ==‘Albert Einstein’:

Take-home

- Homework 1 will be out tomorrow (Friday)!
- Practice Boolean Operations
- http://learnpythonthehardway.org/book/ex28.html
- Read Chapter 5.1-5.6

Next lecture (Monday)

- Nested Conditionals
- For Loops
- What is Module?