Making Decisions: Conditional Execution

Computer Science 111
Boston University

Vahid Azadeh-Ranjbar, Ph.D.

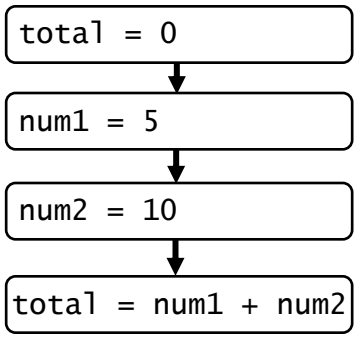
Flow of Control

- 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
```

corresponding flowchart



Conditional Execution

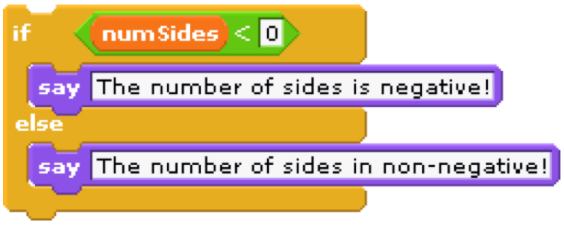
- To solve many types of problems, we need to change the standard flow of control.
- Conditional execution allows your code to *decide* whether to do something, based on some condition.
  - example:

```
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:

```
>>> abs_value(-5)
5
>>> abs_value(10)
10
```

Recall: Conditional Execution in Scratch



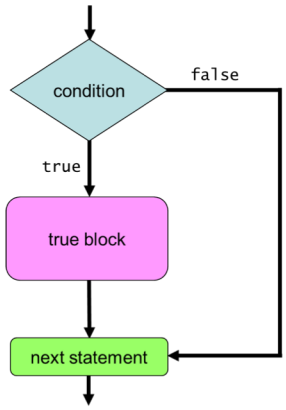
Simple Decisions: `if` Statements

- Syntax:

```
if condition:
    true block
```

where:

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



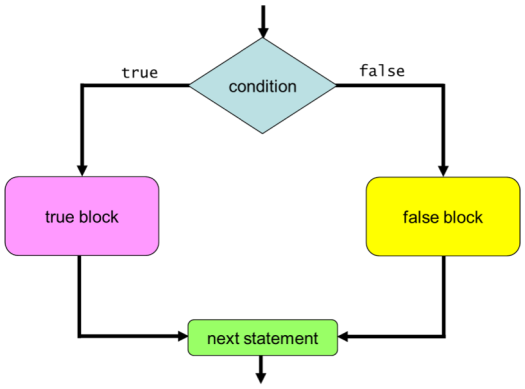
- Example:

```
def abs_value(x):
    if x < 0:
        x = -1 * x     # true block
    return x
```

Two-Way Decisions: `if-else` Statements

- Syntax:

```
if condition:
    true block
else:
    false block
```



- Example:

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'     # true block
    else:
        grade = 'fail'     # false block
    return grade
```

Expressing Simple Conditions

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

operator	name	examples
`<`	less than	`val < 10` `price < 10.99`
`>`	greater than	`num > 60` `state > 'Ohio'`
`<=`	less than or equal to	`average <= 85.8`
`>=`	greater than or equal to	`name >= 'Jones'`
`==` *(don't confuse with =)*	equal to	`total == 10` `letter == 'P'`
`!=`	not equal to	`age != my_age`

Boolean Expressions

- A condition has one of two values: `True` or `False`.

```
>>> 10 < 20
True
>>> "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:

name	example and meaning
`and`	`age >= 18 and age <= 35` `True` *if both conditions are* `True`*, and* `False` *otherwise*

`or`	`age < 3 or age > 65`
	`True` *if one or both of the conditions are* `True`*;*
	`False` *if both conditions are* `False`

`not`	`not (grade > 80)` `True` *if the condition is* `False`*, and* `False` *if it is* `True`

A Word About Blocks

- A block can contain multiple statements.

```
def welcome(class):
    if class == 'frosh':
        print('Welcome to BU!')
        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

Nesting

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

```
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?

```
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:
        print('three')
# program would go here next...
```

A. one

B. two

C. three

D. more than one of the above

E. nothing is output

What does this print? (note the changes!)

```
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!)

```
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!)

```
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!)

```
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

Multi-Way Decisions

- The following function doesn't work.

```
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:

```
>>> letter_grade(95)
'D'
```

Multi-Way Decisions (cont.)

- Here's a fixed version:

```
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:

```
>>> letter_grade(95)
'A'
```

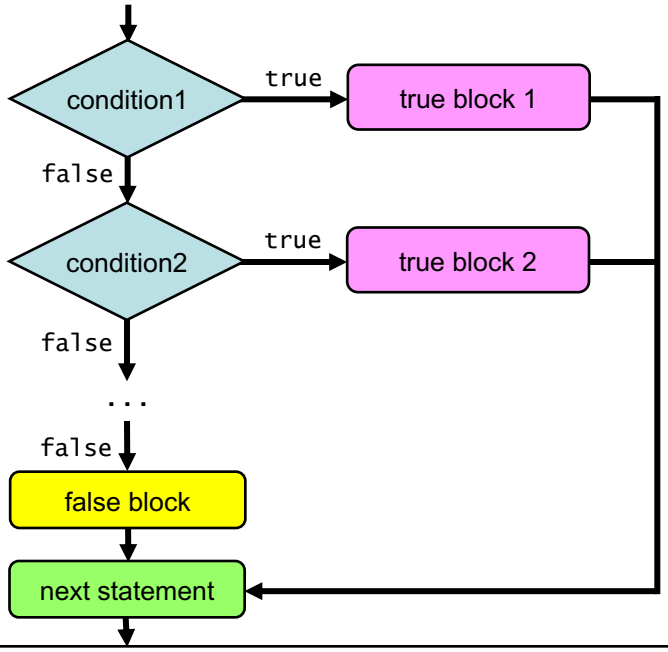
Multi-Way Decisions: `if-elif-else` Statements

- Syntax:

```
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 is true, the false block is executed.

Flowchart for an `if-elif-else` Statement



How many lines does this print?

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

A. 0

B. 1

C. 2

D. 3

E. 4

How many lines does this print?

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

A. 0

B. 1

C. **2**

D. 3

E. 4

How many lines does this print?

```
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

How many lines does this print?

```
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?

a	b
0	5

```
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?

a	b
0	5

```
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`.

Common Mistake When Using and / or

```
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:

```
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 really old.')
```

- How could it be simplified?

Avoid Overly Complicated Code

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

```
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 really old.')
```

- How could it be simplified?

Avoid Overly Complicated Code

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

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

- How could it be simplified?

PS 0 – due Sunday, read instructions carefully!

- Problem 2:

Important notes

- Make sure to limit yourself to the allowed blocks mentioned in Problem 1.
- Here again, when performing movements, you must limit yourself to `[move]`, `[point in direction]`, and `[turn]` blocks. You must **not** add any new `[go to]` blocks besides the one that we gave you in the initial version of the script for Problem 1.
- All of the line segments in both patterns must have a length of 50.
- For full credit, your revised program must **not** have two separate loops – one that draws the square pattern and one that draws the pointy pattern. Rather, you must use a *single* loop that is capable of drawing either pattern, based on the user's input.

  The key to making this work is to take advantage of variables. Set the values of your variable or variables on the basis of the user's input, and then use those variables in the context of a single loop to perform the appropriate actions for the figure that the user chose.
- You may use ***at most one `[if then]` or `[if then else]` statement inside your loop***, and ***it must control at most one or two statements.*** Most of the statements in the loop should be executed regardless of which pattern in being drawn. (Note that you will also need an additional `if` or `if-else` statement ***before the loop*** to process the user's input. That `if` or `if-else` *is* allowed to control more than two statements, but those statements should *not* include the loop used to draw the pattern.)

The Problem Set 0 FAQ includes some additional details about these requirements.

- Submission guidelines:

Important

- It is your responsibility to ensure that the correct version of each file is on Gradescope before the deadline, and that any preliminary Autograder tests have been passed. ***We will not accept any file after the submission window for a given assignment has closed, so please check your submission carefully using the steps outlined above.***
- If you are unable to access Gradescope and there is enough time to do so, wait an hour or two and then try again. If you are unable to submit and it is close to the deadline, email your homework ***before the deadline*** to `cs111-staff@cs.bu.edu`

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade


>>> pass_fail(80)
```

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade


>>> pass_fail(80)
```

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade


>>> pass_fail(80)
```

avg | 80

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade


>>> pass_fail(80)
```

avg: 80

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade


>>> pass_fail(80)
```

avg: 80

grade: 'pass'

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade


>>> pass_fail(80)
```

avg: 80

grade: 'pass'

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade
           'pass'

>>> pass_fail(80)
```

avg: 80

grade: 'pass'

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade
```

avg: 80

grade: 'pass'

```
>>> pass_fail(80)
'pass'
```

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade
```

avg: 80

grade: 'pass'

```
>>> pass_fail(80)
'pass'
```

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade


>>> pass_fail(55)
```

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade


>>> pass_fail(55)
```

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade
```

avg 55

```
>>> pass_fail(55)
```

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade
```

avg 55

```
>>> pass_fail(55)
```

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade
```

avg: 55

grade: 'fail'

```
>>> pass_fail(55)
```

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade
```

avg: 55

grade: 'fail'

```
>>> pass_fail(55)
```

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade
```

avg: 55

grade: 'fail'

```
>>> pass_fail(55)
'fail'
```

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade
```

avg: 55

grade: 'fail'

```
>>> pass_fail(55)
'fail'
```

Tracing Conditional Execution

```
def pass_fail(avg):
    if avg >= 60:
        grade = 'pass'
    else:
        grade = 'fail'
    return grade
```

avg: 55

grade: 'fail'

```
>>> pass_fail(55)
'fail'
```