

# CS 115 Lecture 8

Selection: the if statement

Neil Moore

Department of Computer Science  
University of Kentucky  
Lexington, Kentucky 40506  
neil@cs.uky.edu

24 September 2015

# Selection

Sometime we want to execute code only sometimes.

- Run this code in a certain situation.
  - ▶ How to express “in a certain situation” in code?
- Run this code if this expression is true.
  - ▶ So we'd need true-false expressions.
  - ▶ We mentioned a true-false type in the second week of class.
  - ▶ `bool` (Booleans)

# The boolean type

The type `bool` in Python represents a value that is either true or false.

- Two literals (constant values): `True` and `False`

- ▶ Case-sensitive as always!

- Can have boolean variables:

```
is_finished = False
```

- ▶ Sometimes called **flags** (more on this when we get to loops)

- ...and boolean expressions:

```
is_smallest = number < minimum
```

```
can_run = have_file and is_valid
```

## Naming boolean variables

This isn't a hard-and-fast rule, but try to name boolean variables as a sentence or sentence fragment:

- Is this item selected? – `is_selected`
- Is the user a new user? – `user_is_new` (or `is_user_new`)
- Does the program have an input file? – `have_input_file`
- Does the user want the answer in meters? – `want_meters`

Why `is_selected` and not just `selected`?

- Ambiguous: it could also mean “which item was selected?”

## Type-casting to bool

Most types can be type-cast to bool.

- Usually the meaning is something like “is there anything there?”
- Numbers: 0 (or 0.0) is false, nonzero is true.
  - ▶ Be careful with floats:  $0.3 - 3 * 0.1$  is not exactly zero!
- Strings: the empty string "" is false, anything else is true.
- All graphics shapes are true.
  - ▶ Even `Point(0, 0)`!

# Equality and inequality

Other than literal True and False, the simplest boolean expressions compare the values of two expressions.

- Less than, greater than, ...
- Even simpler: “is equal to” and “is not equal to”.
  - ▶ The equal sign is already taken (for assignment).
  - ▶ So equality testing uses ==  
`logged_in = password == "hunter1"`
    - ★ No spaces between the =!
  - ▶ It's kind of hard to type  $\neq$ , so Python uses != for “is not equal to”:  
`need_plural = quantity != 1`  
`did_fail = actual != expected`
- == compares values, is asks “are they aliases”.

# Comparison

Besides equality and inequality, Python has four more comparison, or **relational**, operators:

- Less than and greater than:

```
score < 60
```

```
damage > hit_points
```

- Less than or equal to (less-equals), greater-equals:

```
students <= seats
```

```
score > 60
```

- The “opposite” of  $<$  is  $\geq$ :  $a < b$  is false if  $a \geq b$  is true.
- Precedence: lower than arithmetic, higher than assignment.

```
need_alert = points + bonus < possible * 0.60
```

is the same as:

```
need_alert = ((points + bonus) < (possible * 0.60))
```

# Relational operators and types

- What type do the relational operators return (i.e. the result)?
  - ▶ `bool`
- What types can be compared with relational operators?
  - ▶ Numbers: `ints` and `floats`.
  - ▶ `str` – what does it mean to compare two strings?
    - ★ “ASCIIbetical order”
    - ★ Like alphabetical order, but considers all characters.
    - ★ Characters are compared by their Unicode value.
      - ▶ `'blu-ray' < 'blue'` because `'-'` comes before `'e'`
    - ★ Uppercase Z comes *before* lowercase a!
- Relational operators cannot mix strings and numbers!
  - ▶ `3 < "Hello"`
    - `TypeError: unorderable types: int() < str()`
  - ▶ It's okay to mix ints and floats, though.



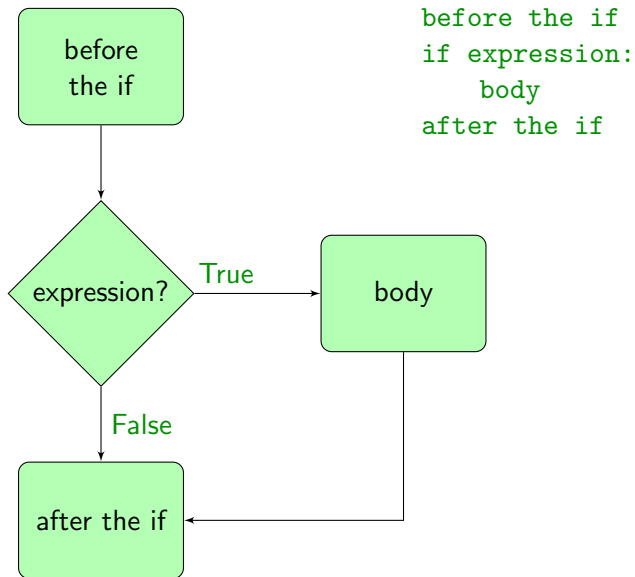
# The `if` statement

Now that we can write some boolean expressions, how do we use those to control whether or not certain code executes?

- Use an **if** statement.
- Syntax:  

```
if expression:  
    body
```
- The expression should evaluate to True or False.
- The body is an indented block of code.
- Semantics: Evaluates the expression.  
Runs the body if it was true.  
Goes on to the line after the body either way.

## Flowchart for if



## Alternatives: else

Commonly we want to **either** do this **or** do that (but not both).

- In Python we can use an **else** block. Syntax:

```
if expression:
```

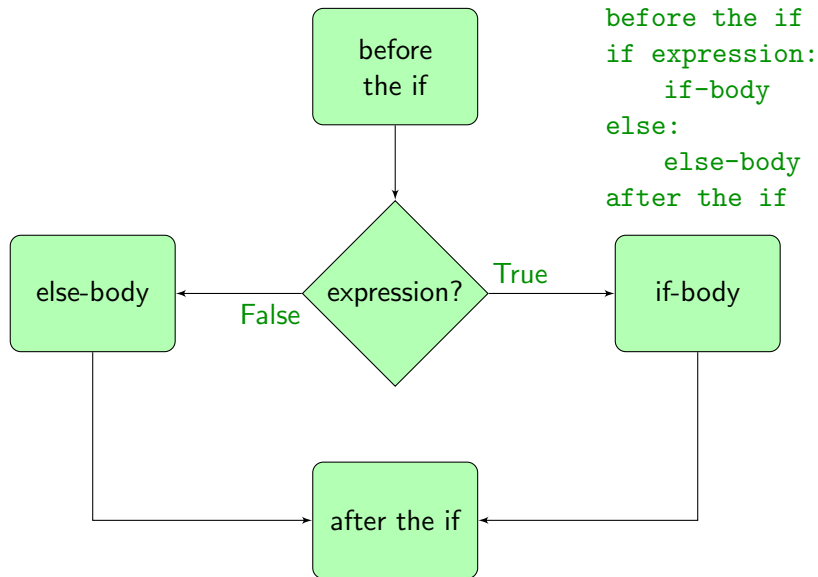
```
    if-body
```

```
else:
```

```
    else-body
```

- ▶ Both bodies are indented blocks.
  - ▶ No expression after “else”!
  - ▶ Can't have an else without an if!
- Semantics:
    - ▶ Evaluates the expression.
    - ▶ If the expression is true, runs the if-body.
    - ▶ Otherwise (it was false), runs the else-body.
    - ▶ Either way, goes on to the line after the else-body.
  - Only use else if there is something to do in the false case.
    - ▶ It's okay not to have one!

## Flowchart for if-else



# Many alternatives

Sometimes there are more than two alternatives.

- Converting a numeric score into a letter grade:
  - ▶ If the score is greater than or equal to 90, print A.
  - ▶ Otherwise, if `score >= 80`, print B.
  - ▶ Otherwise, if `score >= 70`, print C.
  - ▶ And so on...
- We want to run exactly one piece of code.
  - ▶ Even though `95 >= 70`, we don't want 95 to print C too!
  - ▶ First, check if `score >= 90`.
  - ▶ If that was false, check if `score >= 80`.
  - ▶ If that was false too, check if `score >= 70`...
- The order matters!
  - ▶ What would happen if we swapped the order of B and C?
  - ▶ Then we'd never report a B!

## Chained alternatives: `elif`

- Syntax:

```
if expression 1:  
    body 1
```

```
elif expr 2:  
    body 2
```

```
elif expr 3:  
    body 3
```

...

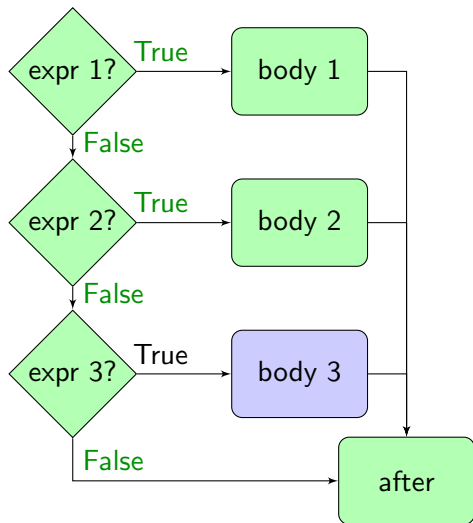
- Semantics:

- ▶ Evaluates expression 1.
- ▶ If expression 1 was true, runs body 1 (and that's all)
- ▶ If expression 1 was false, evaluates expression 2.
- ▶ If expression 2 was true, runs body 2 (and that's all)
- ▶ If expression 2 was false, evaluates expression 3. . .
- ▶ After running at most one body, goes on to the next line.

- Only runs **one** body, or none (the first true expression)

- Each `elif` is followed by an expression.
  - ▶ And a colon.
- Each body is an indented block.
- Can have an `else` block at the very end.
  - ▶ Not required!

## Flowchart for if-elif



```
if expr1:  
    body1  
elif expr2:  
    body2  
elif expr3:  
    body3  
after
```

## Open and closed selection

- If there is an **else**, the selection is **closed**
  - ▶ Meaning exactly one of the bodies will run.
- Otherwise, it is **open**: zero or one bodies will run.
- If the last **elif** is supposed to cover all the remaining cases, prefer **else** instead:

```
if score >= 90:
    grade = 'A':
elif score >= 80:
    grade = 'B':
elif score >= 70:
    grade = 'C':
elif score >= 60:
    grade = 'D':
elif score < 60: else:
    grade = 'E':
```



## When and how to use **elif**

- `divisible.py` `divisible-better.py` `divisible-best.py`
- If you want more than one body to execute, you don't want **elif**.
- Instead, use a sequence of separate ifs.

# Testing ifs

When testing programs with if statements, be sure to consider and test **all** the possible outcomes.

- If your tests never execute a particular line, you don't know if it works!
- For every if or if-else you should have two cases:
  - ▶ One where it is true.
  - ▶ One where it is false—even if there is no else.
- For a chained if-elif, test:
  - ▶ Expression 1 is true.
  - ▶ Expression 1 is false, 2 is true.
  - ▶ Expressions 1 and 2 are false, 3 is true.
  - ▶ ...
  - ▶ All the expressions are false.
  - ▶ If plus  $N$  elifs:  $N + 2$  test cases!

## More testing

- It helps to consider combinations of separate if statements, too.
  - ▶ Especially when they use the same variable(s):

```
if user != "hunter":  
    is_valid = False  
if password != "hedges":  
    is_valid = False
```
- We might have four test cases for these two ifs:
  - ▶ User name right, password right.
  - ▶ User name right, password wrong.
  - ▶ User name wrong, password right.
  - ▶ User name wrong, password wrong.
- Finally, when testing comparisons, check the **boundary** cases:
  - ▶ What if the score is exactly 60.0?
  - ▶ What if the score is 59.9?