

CDS 230

Modeling and Simulation I

Module 3

Control Flow: Comparisons, Logic, and Conditional Statements



Dr. Hamdi Kavak
<http://www.hamdikavak.com>
hkavak@gmu.edu



What have we learned so far?

- Modeling and simulation basics (Week 1)
 - What is a model
 - Why we need models
 - Example models
- Setting up Python environment (Week 2)
 - Anaconda installation
 - Jupyter notebook
 - Cells types
 - Running code
 - Restarting

What have we learned so far?

- Data types
 - Integer
 - Float (i.e., decimal)
 - Complex number
- Variables
- Math functions
 - abs, round, exp, ..., cos, sin, ...
- Some simple models
 - Linear motion
 - Free fall

What will we learn this week?

- Control flow (today – Sep 8)
 - Comparisons, Logic, and Conditional Statements
 - How to tell Python to make comparisons and control the flow of your code
- Strings (Wed – Sep 13)
 - Dealing with text type of data (we dealt with numbers so far)

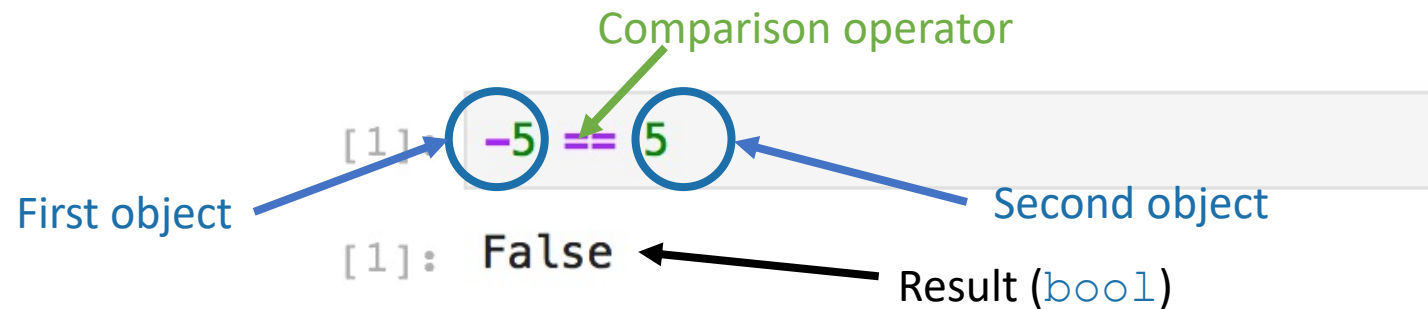
Comparison in the real-world

- We do comparisons all the time. For instance:
- Compare prices from different stores..
- Check if the weather is above certain degrees..
- Is the gas in my car near empty..
- What is the best class you're taking this semester ..
- Any other examples?

Since models help us represent and study something in the real-world, we want to be able to code such comparison cases

Comparison operators

- In Python, objects can be compared via comparison operators



- `==` checks if two objects are equal, but there are other operators
 - `!=` not equal to
 - `>` greater than
 - `>=` greater than or equal to
 - `<` less than
 - `<=` less than or equal to

Comparison tips

- Comparing `integers` with `floats`

```
[4]: 3 >= 2.718281828459045 # e number
```

```
[4]: True
```

- How about `=` vs. `==` ?

```
[5]: x=5
```

```
[6]: x==5
```

- Float comparison:

```
[9]: a = 0.1  
     b = 0.2  
     c = a + b  
     print(c == 0.3)
```

Logic operators

- Comparisons can be compounded using logic operators: `not`, `and`, `or`
- Logic operators have lower precedence than comparison operators unless you use parentheses.

- How they work

- `not` operator negates a `bool` value.
- `and` operator is placed between two `bool` values and returns `True` only when both values are `True` otherwise, it returns `False`.
- `or` operator is placed between two `bool` values and returns `True` when at least one value is `True` otherwise, it returns `False`.

[12]: `not 4 > 2`

[14]: `1 == 1 and 4 > 2`

[15]: `import math
math.pi > math.e or 1 < 2`

Higher
precedence

Lower
precedence

Logic operators (code)

```
[ ]: not 5 > 3 and 6 + 1 > 7
```

```
[ ]: 3 * 5 < 15 or math.sqrt(4) == 2
```

```
[ ]: not 4 > 3.99999 or 5 % 2 == 1
```

```
[ ]: not (4 > 3.99999 or 5 % 2 == 1)
```

```
[ ]: not 4 > 3.99999 and 5 % 2 == 1
```

Some detailed rules

- Numeric to `bool`
 - `bool()` function can cast numbers to Boolean values (`True` or `False`) like `int(5.4)` casting floats to integers.
 - Rule is simple: if you pass 0 to `bool()` function, it will return `False`; otherwise it will return `True`.
- `None` represents an undefined value
 - Used for cases where no value is possible or relevant (e.g., to avoid default initial value assignments).
 - `bool(None)` returns `False`;
 - How to check if a variable has `None` as its value? use `is` or `is not`

Some detailed rules (code)

```
[ ]: bool(0)
```

```
[ ]: bool(0.0)
```

```
[ ]: bool(complex(0,0))
```

```
[ ]: bool(9999)
```

```
[ ]: bool(-9999)
```

```
[ ]: bool(0.00000001)
```

```
[ ]: bool(complex(0,1))
```

```
[ ]: bool(None)
```


```
[ ]: bool(not None)
```

```
[ ]: not not bool(None)
```

```
[ ]: x = None  
print (x is None)  
x = 4  
print (x is None)
```

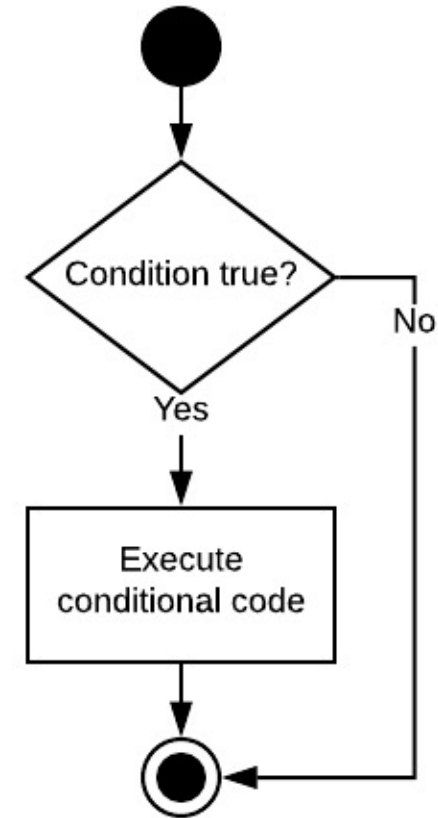
AI systems...



Source: <http://www.commitstrip.com/en/2017/06/07/ai-inside/> 

Conditional statements

- Allows program to execute or skip different parts of the code dynamically.
- Comparison operators are used here
- Main conditional statements
 - `if`
 - `else`
 - `elif`
 - No `switch-case` support in Python 😞



if

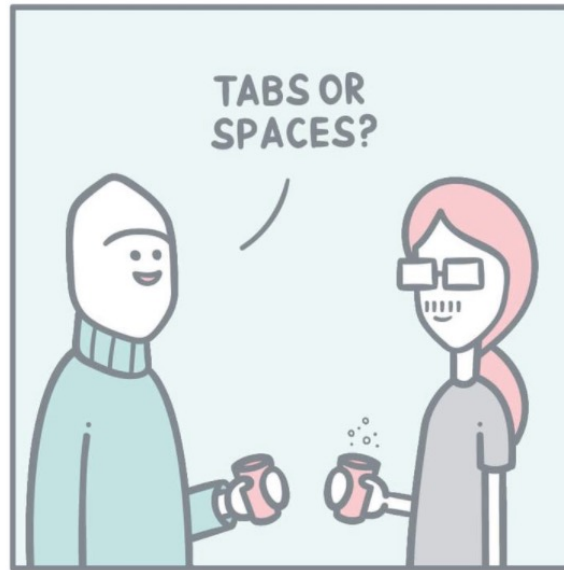
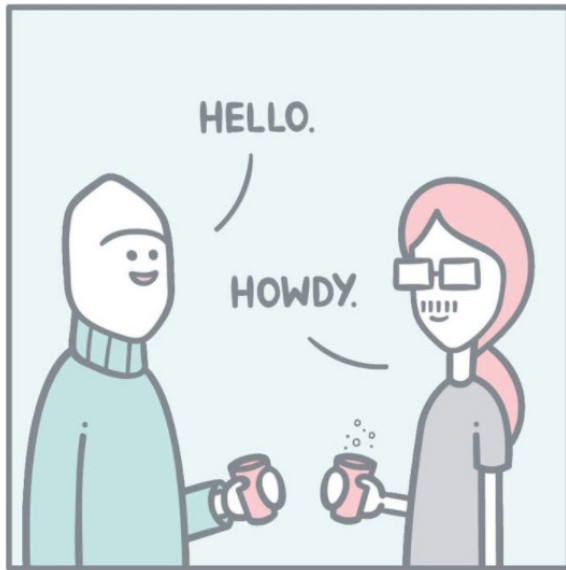
```
1  if c > 5:  
2      command 1  
3      command 2
```

```
1  x = 6  
2  if x > 4:  
3      print('Yes')
```

Yes

- > < >= <= == !=
- Does not require parenthesis
- Colon after the **if** statement
- Indentation for following commands.
- If statement checks the result of the comparison statement before the colon

Don't mix tabs and spaces for indentations



Source: <https://crystallize.com/comics/tabs-vs-spaces>

Indentations

Python requires proper indentation.

No `BEGIN` or `END`

No `{` or `}`

Indentations indicate what is inside or outside of an `if` statement

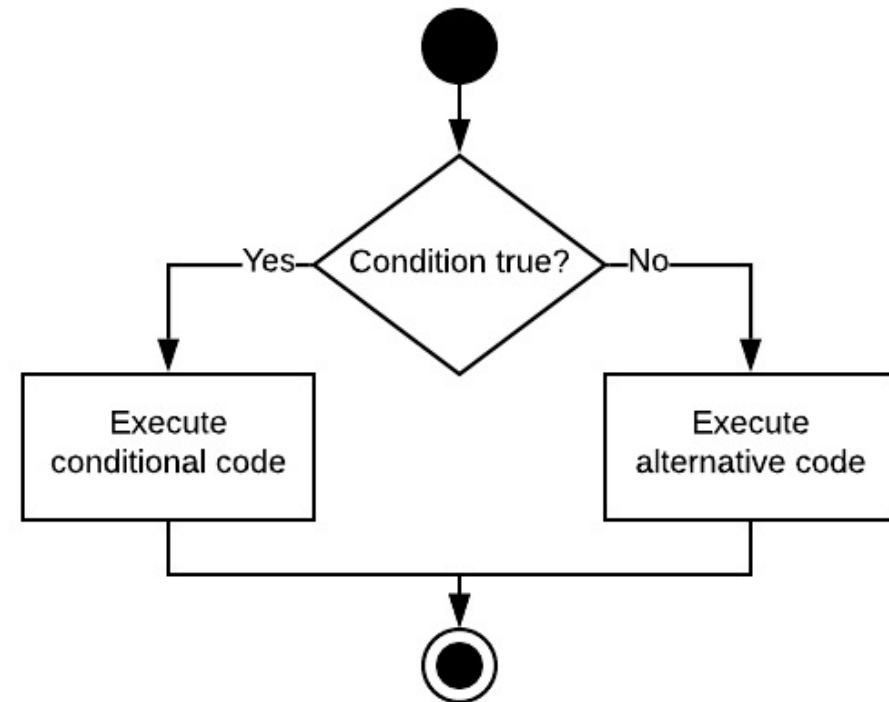
```
1 x = 6
2 if x > 4:
3     print('Yes')
4     print('More Yes')
```

Yes

More Yes

if - else

- If the condition is true, execute the conditional code (**left path**).
- If the condition is false, execute the alternative code (**right path**).



if - else

```
1 x = 2
2 if x > 4:
3     print('Yes')
4     print('More Yes')
5 else:
6     print('No')
```

No

Else statement used when **if** statement is false.
Ends with a colon

elif

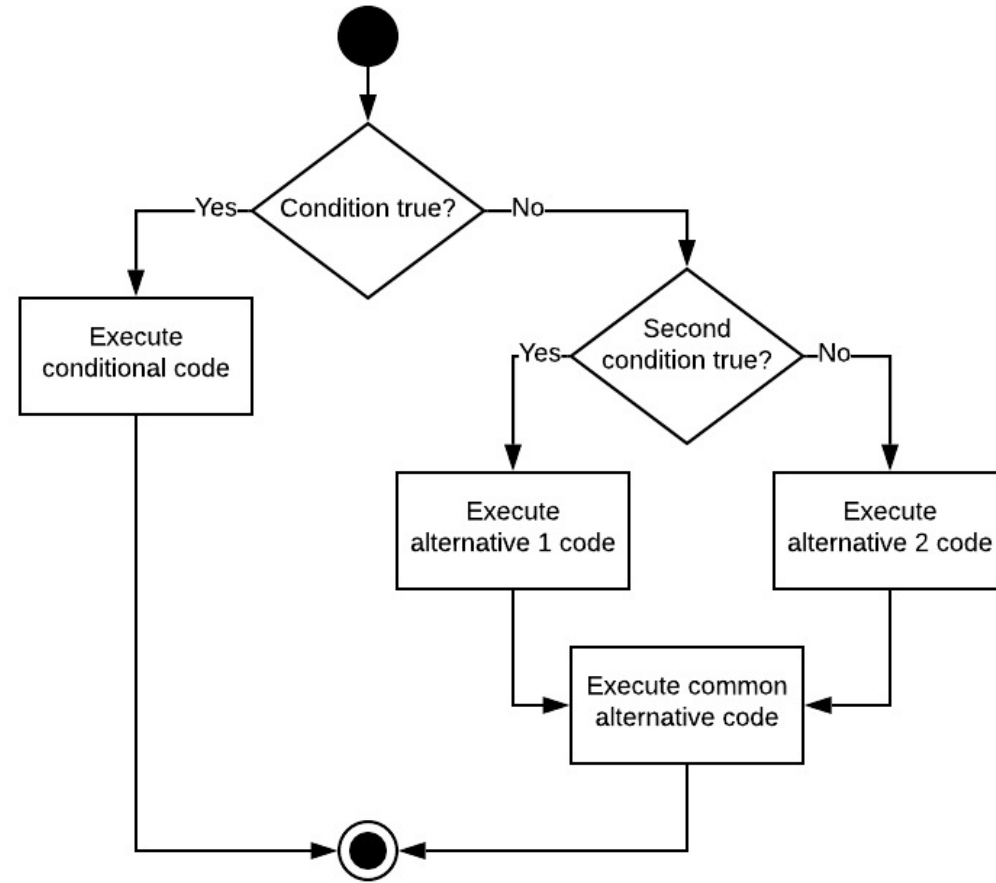
- Equals to `else if`.

```
[ ]: x = 5
      y = 7
      if x < 5:
          print('X is less than 5')
      elif x >=5 and y < 10:
          print('x is greater than or equal to 5, but y is less than 10')
      else:
          print('Neither of the two above conditions hold')
```

What if we need more than one
conditions to check?

Nested conditional statements

- Multiple `if`, `else`, and `elif` statements one inside another.
- Indentation may become tricky to handle



Nested conditional statements (code)

```
[ ]: x = 5
      y = 7
      z = None
      if x < 5:
          print('X is less than 5')
      elif x >=5 and y < 10:
          print('x is greater than or equal to 5, but y is less than 10')
          if z is None:
              print('z is None')
      else:
          print('Neither of the two above conditions hold')
```