Section 001 CDS 411 *Modeling and Simulation II*

Python fundamentals and scientific computing with Python

Class 4

Control flow

Loops

- What are loops and iteration?
- Basic for loops with range()
- For loop over list with enumeration
- DON'T USE COUNTER VARIABLES
- Looping over dictionaries
- List comprehensions, or, "the in-line for loop"

List comprehensions

Assume we have a list within a list:

```
mylist_2d = [[0, 2], [2, 4], [4, 6], [6, 8]]
```

List comprehensions make it simple to extract the values within the first and second *columns*:

```
mylist_col_one = [value[0] for value in mylist_2d]
mylist_col_two = [value[1] for value in mylist_2d]
print("Column 1: {0}\nColumn 2: {1}".format(mylist_col_one,
                                            mylist_col_two))
```

```
## Column 1: [0, 2, 4, 6]
## Column 2: [2, 4, 6, 8]
```

List comprehensions

Alternative would be to use a `for` loop:

```
mylist_col_one = []
mylist_col_two = []
for value in mylist_2d:
    mylist_col_one.append(value[0])
    mylist_col_two.append(value[1])

print("Column 1: {0}\nColumn 2: {1}".format(mylist_col_one,
                                             mylist_col_two))
```

```
## Column 1: [0, 2, 4, 6]
## Column 2: [2, 4, 6, 8]
```

Boolean logic: Comparisons

Operation	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal
is	object identity
is not	negated object identity

```
print(1 < 2,
      True == False,
      True is False,
      sep = "\n")
```

```
## True
## False
## False
```

```
x = None
print(x == None,  #  not "Pythonic"
      x is None,  # "Pythonic"
      sep = "\n")
```

```
## True
## True
```

Boolean logic: Operators and chaining

Operators: `and`, `or`, `not`

Operation	Result
`x or y`	if x is false, then y, else x
`x and y`	if x is false, then x, else y
`not x`	if x is false, then `True`, else `False`

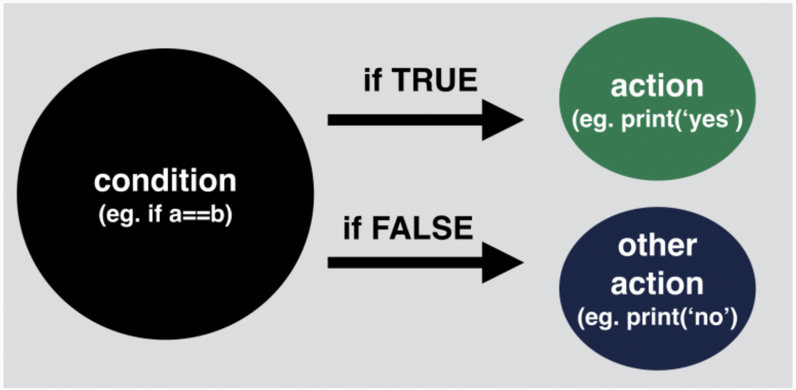
Chaining

- Two ways to write mathematical statements like $a \leq b < c$.
- More complex statements should be enclosed in parentheses and connected with operators

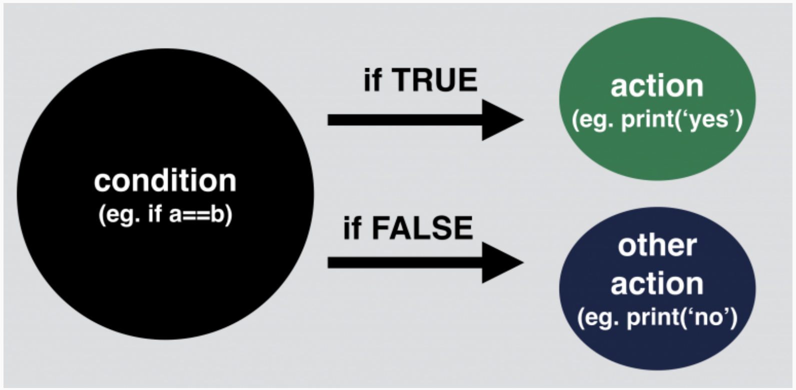
```
a, b, c = (3, 4, 5)
print(a <= b < c,
      (a <= b) and (b < c),
      sep = "\n")
```

```
## True
## True
```

Branching with if statements



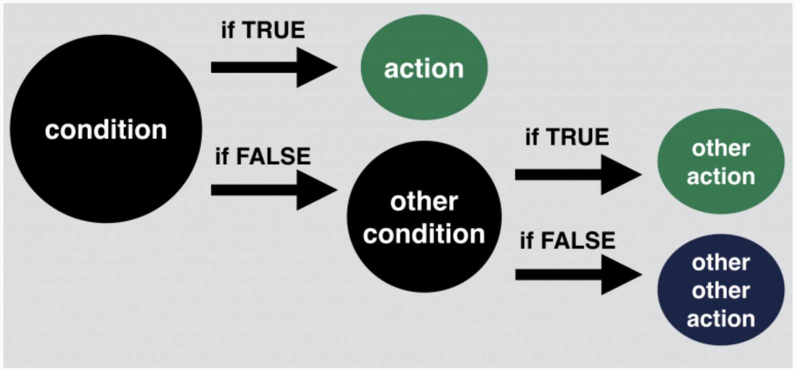
Branching with if statements



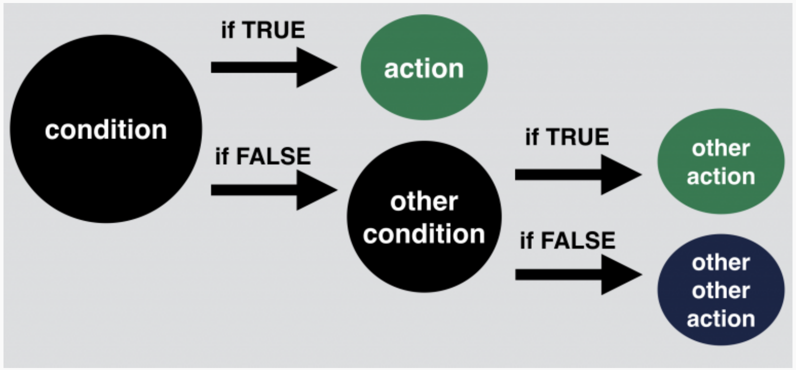
```
a = 10
b = 20
c = 30

if (a + b) / c == 1 and c - b - a == 0:
    print('yes')

else:
    print('no')
```

```
## yes
```

Branching with if statements



Branching with if statements



```
a = 10
b = 11
c = 10

if a == b:
    print('first condition is true')

elif a == c:
    print('second condition is true')

else:
    print('nothing is true. existence is pain.')
```

```
## second condition is true
```

Whitespace in Python

- How Python uses whitespace
- When it's mandatory
- When it's optional, but recommended

User-defined functions

Functions: What are they?

- You've used them already, you just haven't made your own.
- Why bother?
- Don't Repeat Yourself (DRY) and building with legos
- Ideal use case: small re-usable pieces that do one thing.
- Basic idea: You have a set of instructions that take inputs and steps through the instructions. You may or may not get an output, depending on what you want to do.
- Every modern language uses them, even Matlab!

```matlab
% Example of a function definition in Matlab

function m = avg(x,n)
m = sum(x)/n;
end
```

Structure of a Python function

```python
def your_awesome_function(input1, input2, input3="default"):
    """The docstring that explains what your function does.

    Information about the inputs usually follows.
    """
    # A list of commands, just like you would write in a
    # procedural Python program.
    if input3 == "default:
        a = input1 + input2

    else:
        a = 0

    # How to return the value of a
    return a
```

Getting used to making functions

```python
a = 10
b = 11
c = 10

if a == b:
    print('first condition is true')

elif a == c:
    print('second condition is true')

else:
    print('nothing is true. existence is pain.')
```

Getting used to making functions

```python
def test_my_inputs(a, b, c):
    if a == b:
        message = 'first condition is true'

    elif a == c:
        message = 'second condition is true'

    else:
        message = 'nothing is true. existence is pain.'

    return message


print(test_my_inputs(10, 11, 10))
```

```
## second condition is true
```

Scientific computing with the `numpy` package

Building up n-dimensional arrays

Import numpy package with alias `np`:

```
import numpy as np
```

```
mylist_2d = [[87, 84, 91],
             [19, 64, 25],
             [24, 95, 9]]
mymatrix_2d = np.array(mylist_2d)
print(mymatrix_2d)
```

```
## [[87 84 91]
##  [19 64 25]
##  [24 95  9]]
```

Numpy demos

Demo

Credits

License

Acknowledgments

Source for If statement examples: https://data36.com/python-if-statements-explained-data-science-basics/