UNIVERSITÄT BIELEFELD
Faculty of Technology

Programming

Data types, mutability, conditions & comparisons

Luna Pianesi

Faculty of Technology, Bielefeld University

Recap

During our last lecture we talked about:

- Computer architecture
- Overview of Python
- Anaconda, Qt Console & Jupyter
- Pythons basics

Recap

Arithmetic in Python

Numeric types:

- Integer: `int` `42`
- Real valued numbers: `float` `42.0`
- Complex numbers: `complex` `42+0j`

Operators

- Addition and subtraction `+ -`
- Multiplication and division `* / // %`
- Exponentiation `**`

Variables

Variable assignment

- `a = 42`
- `b = a - 6.0`

`type(«name of the variable»)`: returns type of variable

Libraries

Importing libraries

- `import numpy as np`
- `import matplotlib.pyplot as plt`

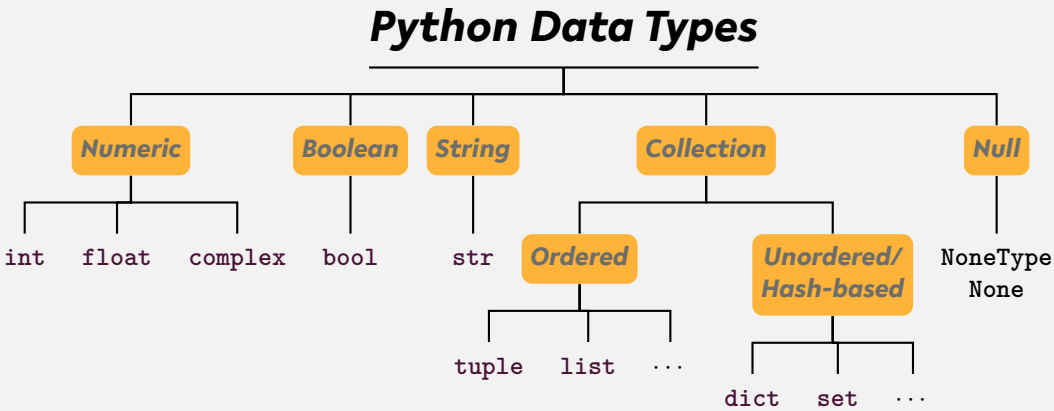
`import` «`name of the library`»`as` «`alias`»: loads the requested library under the alias' name

Data Types & Mutability

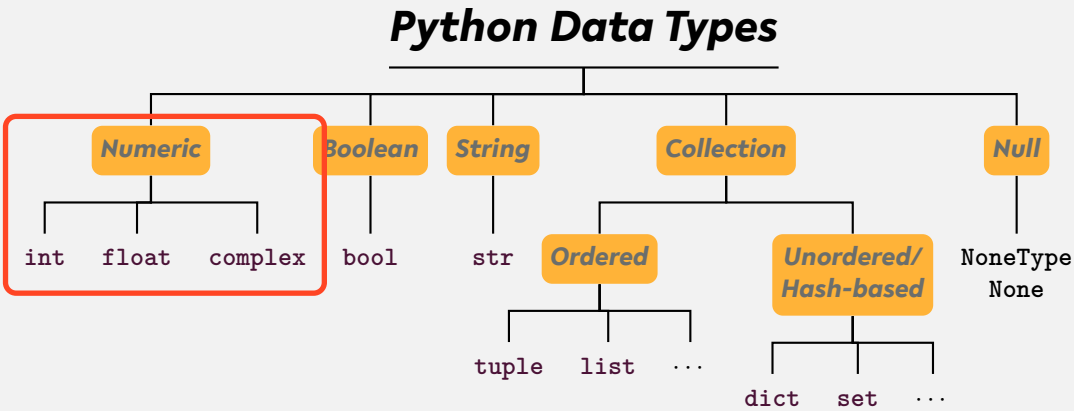
Evaluation Order

Conditions

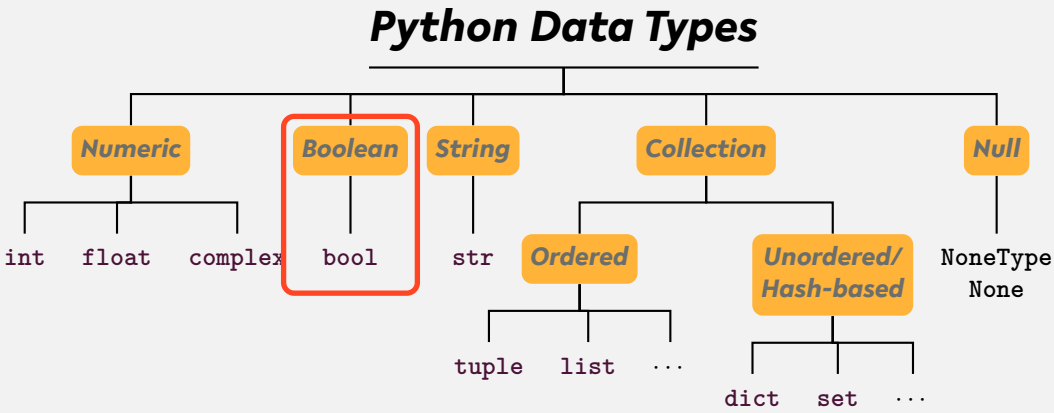
Comparisons



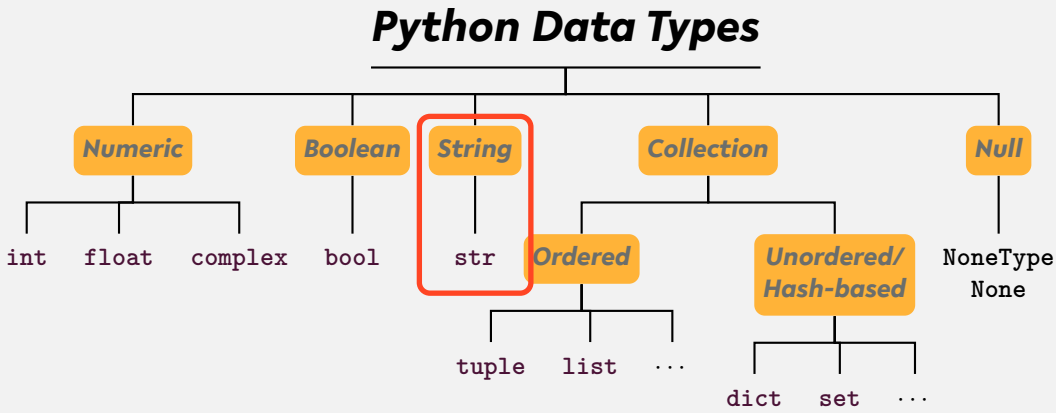
... and user-defined types



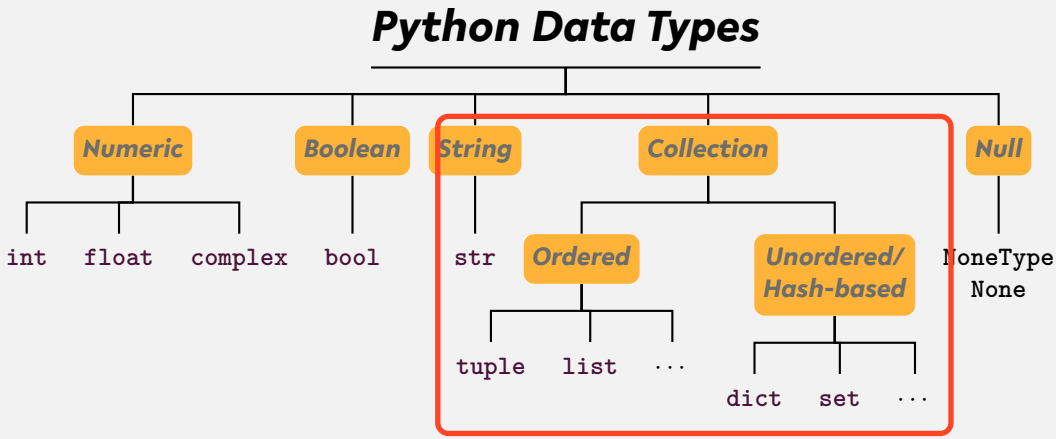
... and user-defined types



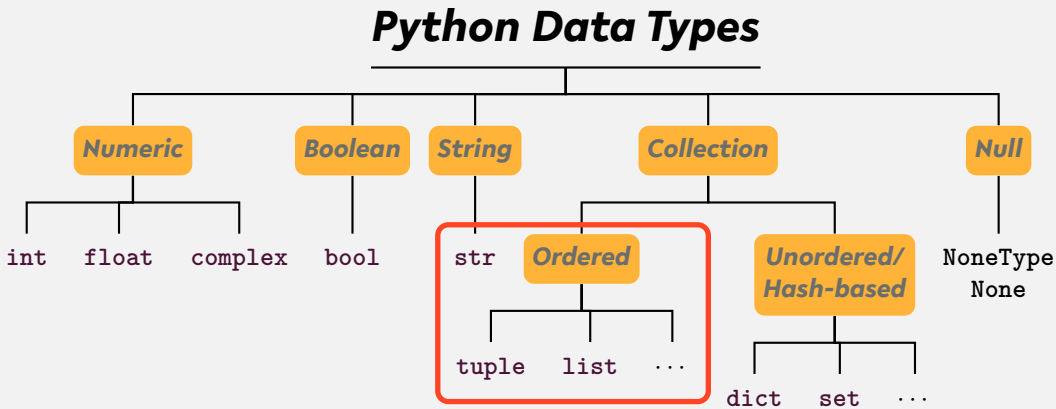
... and user-defined types



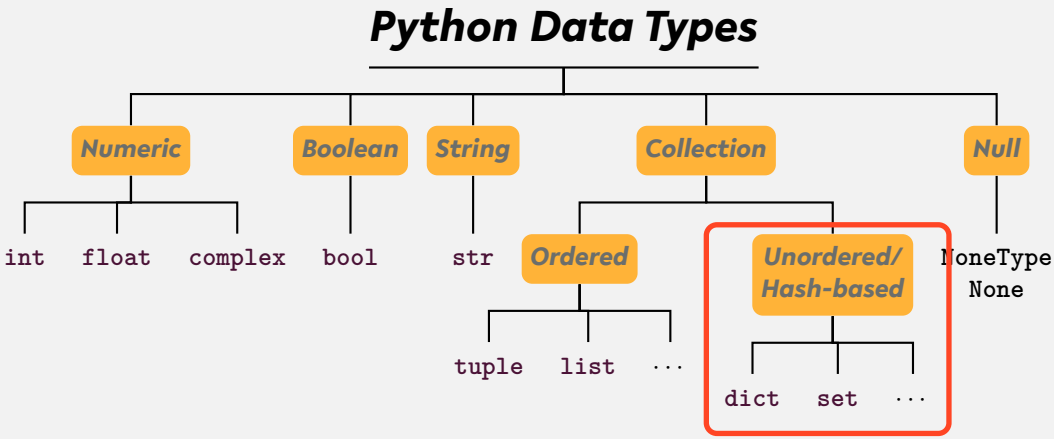
... and user-defined types



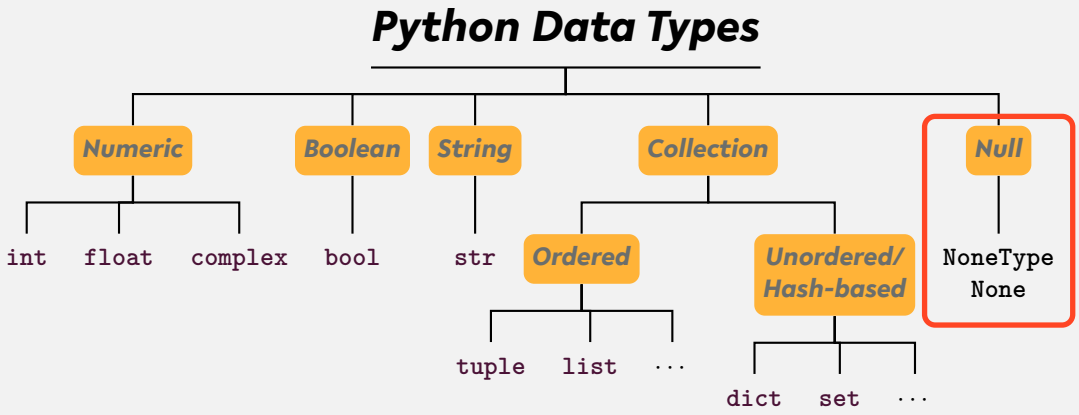
... and user-defined types



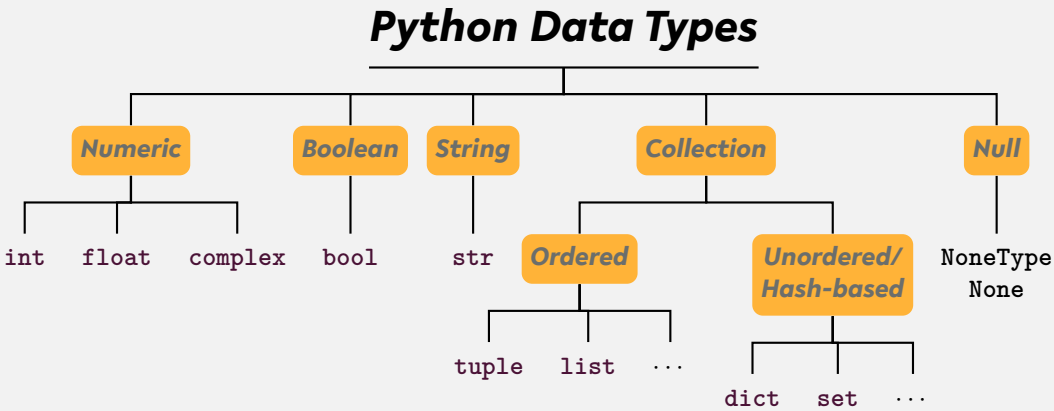
... and user-defined types



... and user-defined types

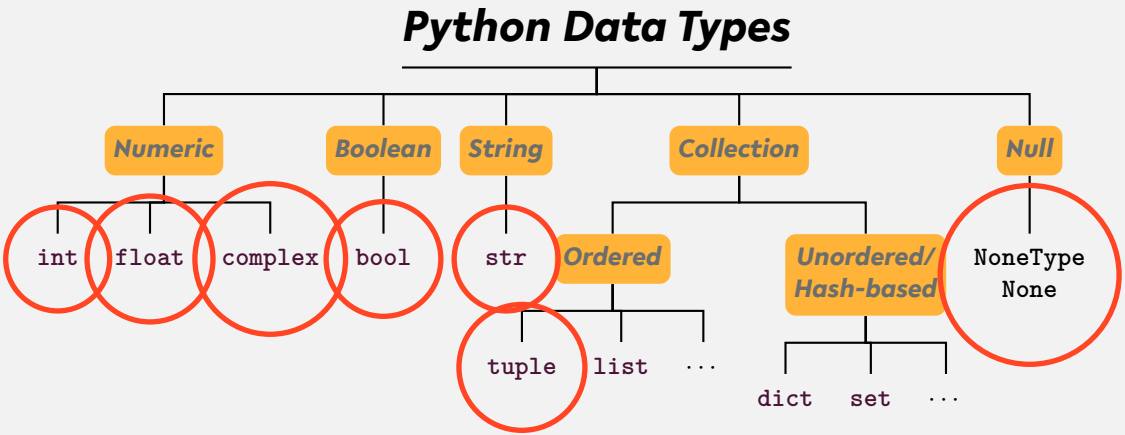


... and user-defined types



... and user-defined types

We differentiate between ***type*** and ***instance***!



... and user-defined types

Instances of certain types are ***immutable*** , i.e., ***cannot be changed after creation***

Types, instances, variables

We differentiate between ***type***, ***instance***, and ***variable***!

```
1 a = list()
2 b = a
3 b.append(1)
4 b = 'this␣is␣a␣string'
```

Lines 1-3: Instance of type `list` is assigned to variables `a` and `b`.

Line 4: Variable `b` refers now to a new string instance

Memory address: `id`

- Every instance has a unique address in memory
- `id(x)`: memory address of instance of `x`
- `x`, `y` reference the same instance if and only if `x` is equal to `y`.

Numeric types - immutable

`int()`, `float()`, `complex()`

- Instantiation: `x = 42`, `x = 42.0` or `x = 42+0j`
- Length: NO `len()` function
- Access: NO access
- Existence: NO existence
- Frequency: NO frequency

Boolean - immutable

`bool()`

- Instantiation: `val = True` or `val = False` (capital)
- Length: NO `len()` function
- Access: NO access
- Existence: NO existence
- Frequency: NO frequency

String - immutable

`str()`

- Instantiation: `s = 'a new string'` or `s = "a new string"`
- Length: `len(s)`
- Access:
  - First: `s[0]`
  - Slice: `s[1:3]`
  - Last: `s[-1]`
- Existence: `'n' in s` or `'new' in s`
- Frequency: `s.count('new')`

Tuple - immutable

`tuple()`

- Instantiation: `t = (1, 'two', 3.0)`
- Length: `len(t)`
- Access:
  - First: `t[0]`
  - Slice: `t[1:3]`
  - Last: `t[-1]`
- Existence: `'two' in t` or `3 in t`
- Frequency: `s.count(3.0)`

List - mutable

`list()`

- Instantiation: `l = [1, 2, 3]`
- Length: `len(l)`
- Add elements: `l.append("content")`
- Access:
  - first: `l[0]`
  - slice: `l[1:3]`
  - last: `l[-1]`
- Existence: `2 in l`
- Location: `l.index(3)`

Dictionary - mutable

`dict()`

- Instantiation: `d = dict()`, `d = {'x': 1, 'y': 2}`, ...
- Length: `len(d)`
- Add elements: `d['a'] = 'four'`
- Access: `d['a']`
- Existence: `'a' in d`

Set - mutable

`str()`

- instantiation: `s = set()`, `s = 1, 2, 15.0, 0`, ...
- Length: `len(s)`
- Add elements: `s.add(3)`
- Access: NO access
- Existence: `15.0 in s`

None - immutable

`None`

- instantiation: `var = None` (capital)
- Length: NO length
- Access: NO access
- Existence: NO existence

Type conversion

- Python is smart in converting basic data types
- `int(.)`, `float(.)`, `tuple(.)`, …
- Everything evaluates to a Boolean value
  - Boolean conversion is even performed implicitly

Quiz

Which of the following are valid ways to specify strings in Python:

```
"test"      'test'      "foo'bar"      'foo'bar'
```

True or *false*?

- “In a dictionary, values are accessed by their position.”
- “A variable can only reference a single instance at a time.”
- “Data types are placeholders for instances.”
- “Instances are placeholders for data types.”
- “The expression `bool('None')` evaluates to `False`.”

Quiz

Which of the following are valid ways to specify strings in Python:

`"test"` ✔ `'test"` `"foo'bar"` ✔ `'foo'bar'`

True or *false*?

- “In a dictionary, values are accessed by their position.” false
- “A variable can only reference a single instance at a time.” true
- “Data types are placeholders for instances.” false
- “Instances are placeholders for data types.” false
- “The expression `bool('None')` evaluates to `False`.” false


Data Types &
Mutability
Evaluation
Order
Conditions
Comparisons

UNIVERSITÄT BIELEFELD

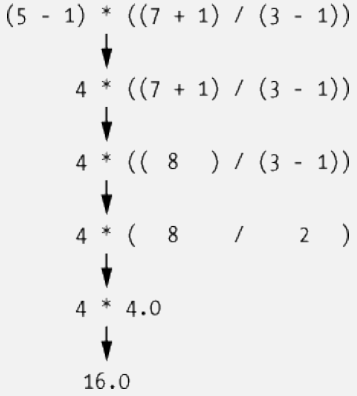
Operator precedence

Parentheses	`( ... )`
Exponents	`**`
Multiplication and ***D***ivision	`* / // %`
Addition and ***S***ubstraction	`+ -`

https://en.wikibooks.org/wiki/Python_Programming/Basic_Math

Expression evaluation

Evaluation: operator precedence + ***left-to-right***



Automate the Boring Stuff with Python - Al Sweigart (CC-BY-NC-SA 3.0) chapter 1, figure 1-1, `https://automatetheboringstuff.com/chapter1/`

Operator Precedence

low

high

Operator	Description
`=, +=, -=, |=, ...`	Assignment expression
`lambda`	Lambda expression
`if - else`	Conditional expression
`or`	Boolean OR
`and`	Boolean AND
`not x`	Boolean NOT
`in, not in, is, is not, <, <=, >, >=, !=, ==`	Comparisons, including membership tests and identity tests
`|`	Bitwise OR
`^`	Bitwise XOR
`&`	Bitwise AND
`<<, >>`	Shifts
`+, -`	Addition and subtraction
`*, @, /, //, %`	Multiplication, matrix multiplication, division, floor division, remainder 5
`+x, -x, ~x`	Positive, negative, bitwise NOT
`**`	Exponentiation 6
`await x`	Await expression
`x[index], x[index:index], x(arguments...), x.attribute`	Subscription, slicing, call, attribute reference
`(expressions...), [expressions...], key: value..., expressions...`	Binding or parenthesized expression, list display, dictionary display, set display

Quiz

- What is the value of the expression `1 + 2 ** 3 * 4`?
- Which of the following operators has the lowest precedence?

  `and` `+` `**` `%` `not`

- Which operation of the expression `'Tiger'[4] + 'oa'* 4 + 'r'` is executed first?

  `'oa' * 4` `'Tiger'[4] + 'oa'` `'Tiger'[4]` `4 + 'r'`

source (in part): `https://realpython.com/quizzes`

Quiz

- What is the value of the expression `1 + 2 ** 3 * 4`? 33
- Which of the following operators has the lowest precedence?

  `and` ✔ `+` `**` `%` `not`

- Which operation of the expression `'Tiger'[4] + 'oa'* 4 + 'r'` is executed first?

  `'oa' * 4` `'Tiger'[4] + 'oa'` `'Tiger'[4]` ✔ `4 + 'r'`

source (in part): `https://realpython.com/quizzes`

Data Types & Mutability

Evaluation Order

Conditions

Comparisons

Conditional statements: `if/else` *clause*

`if` «*Boolean expression*»:
␣␣␣␣«*statement*»

⚠ Mind the indentation!

OR

`if` «*Boolean expression*»:
␣␣␣␣«*statement*»
`else`:
␣␣␣␣«*alternative statement*»

Conditional statements: `if/else`

```
1 a = True
2 if a:
3     print('a␣is␣True')
4
5     if 'this␣is␣a␣text':
6         print('another␣true␣statement')
```

Conditional statements: `if/else`

```
1 a = True
2 if a:
3     print('a␣is␣True')
4 else:
5     print('a␣is␣False')
```

Data Types & Mutability

Evaluation Order

Conditions

Comparisons

Boolean operators and comparisons

Elementary logic: `and`, `or`, `not`

Variables		Boolean expression		
`a`	`b`	`not a`	`a and b`	`a or b`
`False`	`False`	`True`	`False`	`False`
`False`	`True`	`True`	`False`	`True`
`True`	`False`	`False`	`False`	`True`
`True`	`True`	`False`	`True`	`True`

Comparisons: Operators

- `==` “is equal/equivalent to”
- `!=` “is not equal/equivalent to”
- `>` “is larger than”
- `<` “is is smaller than”
- `>=` “is larger or equal to”
- `<=` “is smaller or equal to”
- `is` “is identical instance of”
- `is not` “is not identical instance of”
- `in` “is contained in collection”
- `not in` “is not contained in collection”

Conditional execution based on comparisons

```
1 a = 4.0
2 b = 2.0
3 if not a > b:
4     print('true␣statement!')
```

Conditional execution based on comparisons

```
1 a = 'this␣is␣a␣text'
2 b = 'this␣is␣a␣text'
3 if a >= b:
4     print('true␣statement!')
```

Conditional execution based on comparisons

```
1 a = 'this␣is␣a␣text'
2 if a:
3     print('true␣statement!')
```

Conditional execution based on comparisons

```
1 a = list()
2 b = list()
3 if a is b:
4     print('true␣statement!')
```

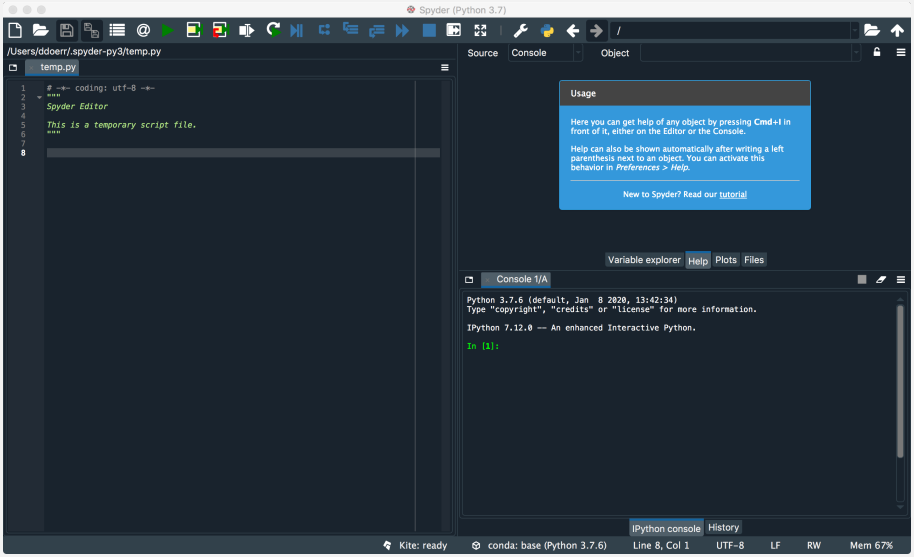
Conditional execution based on comparisons

```
1 a = list()
2 b = list()
3 if a == b:
4     print('true␣statement!')
```

Conditional execution based on comparisons

```
a = list()
b = 1
if b in a:
    print('b␣is␣contained␣in␣collection␣a')
else:
    print('b␣is␣not␣contained␣in␣collection␣a')
```

Spyder


Spyder (Python 3.7)
/Users/ddoerr/.spyder-py3/temp.py
temp.py
-*- coding: utf-8 -*-
"""
Spyder Editor
This is a temporary script file.
"""
Source
Console
Object
Usage
Here you can get help of any object by pressing Cmd+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.
New to Spyder? Read our tutorial
Variable explorer
Help
Plots
Files
Console 1/A
Python 3.7.6 (default, Jan 8 2020, 13:42:34)
Type "copyright", "credits" or "license" for more information.
IPython 7.12.0 -- An enhanced Interactive Python.
In [1]:
IPython console
History
Kite: ready
conda: base (Python 3.7.6)
Line 8, Col 1
UTF-8
LF
RW
Mem 67%

Recap

Summary

- Python data types: `int`, `float`, `str`, `tuple`, `list`, `dict`, …
- Operator precedence
- `if`/`else` clause
- Comparison operators: `==`, `!=`, `>`, `<`, `is`, `in`, …

What comes next?

- Familiarize yourself with Spyder
- Loops (for loops and while loops)
- Write your first program!
- Due date for this week's exercises is ***Saturday, October 30, 2024***.

Next lecture: For loops & while loops …