

Programming

Programming & Python Basics

Daniel Dörr

Faculty of Technology, Bielefeld University

```
332
333
334     if extrapolate is None:
335         extrapolate = self.extrapolate
336     x = np.asarray(x)
337     x_shape, x_ndim = x.shape, x.ndim
338     x = np.ascontiguousarray(x.ravel(), dtype=np
339
340     # With periodic extrapolation we map x to the
341     # [self.t[k], self.t[n]].
342     if extrapolate == 'periodic':
343         n = self.t.size - self.k - 1
344         x = self.t[self.k] + (x - self.t[self.k]) *
345         extrapolate = False
346
347     out = np.empty((len(x), prod(self.c.shape[1:])),
348                   dtype=self._evaluate(x, nu, extrapolate, out)
349                   self._evaluate_spline(self.t, self.c.reshape(self.c
350                   out = out.reshape(x_shape + self.c.shape[1:]))
351
352     if self.axis != 0:
353         # transpose to move the calculated values to t
354         l = list(range(out.ndim))
355         l = l[x_ndim:x_ndim+self.axis] + l[:x_ndim] +
356         out = out.transpose(l)
357     return out
358
359 def _evaluate(self, xp, nu, extrapolate, out):
360     _bspl.evaluate_spline(self.t, self.c.reshape(self.c
361     self.k, xp, nu, extrapolate, out)
362
363 def _ensure_c_contiguous(self):
364     """
365     c and t may be modified by the user. The Cython code
366     c and t are C contiguous.
367     """
368     if not self.c.flags.c_contiguous:
369         self.c = np.ascontiguousarray(self.c)
370     if not self.t.flags.c_contiguous:
371         self.t = np.ascontiguousarray(self.t)
```

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

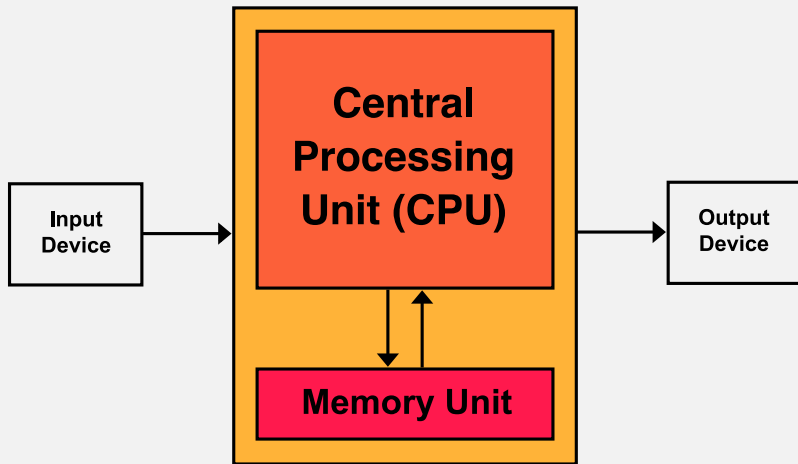
**Programming
Basics**

**Data Types &
Mutability**

**Evaluation Or-
der**

**Conditions &
Comparisons**

Computer architecture



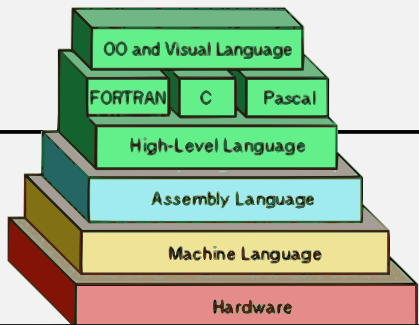
Python compiler: reference implementation

High level language

- Easy to understand

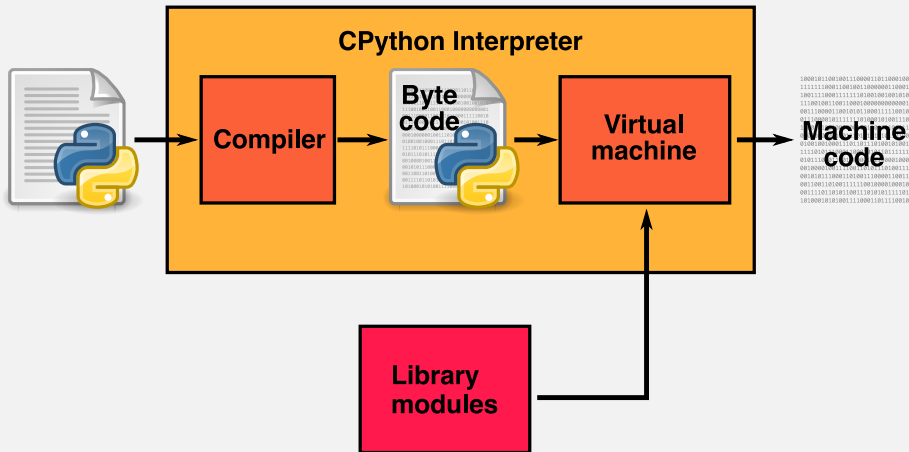
Low level language

- Machine code
(processor instructions
and binary code)



Source: <https://thebittheories.com>

Python compiler: reference implementation



Quiz

True or false?

- ❖ “The three parts of the Von Neumann computer architecture are: Processor, RAM & Hard Disk.”
- ❖ “The CPython interpreter converts Python code to the language that the computer’s hardware understands.”

When you try to run Python scripts, a multi-step process begins. In this process the interpreter performs three steps:

1. Ship off the code for execution.
2. Process the statements of your script in a sequential fashion.
3. Compile the source code to an intermediate format known as bytecode.

Identify the correct order of these steps:

1 → 3 → 2

2 → 3 → 1

1 → 2 → 3

3 → 2 → 1

source: <https://realpython.com/quizzes>

Quiz

True or false?

- ❖ “The three parts of the Von Neumann computer architecture are: Processor, RAM & Hard Disk.” false
- ❖ “The CPython interpreter converts Python code to the language that the computer’s hardware understands.” true

When you try to run Python scripts, a multi-step process begins. In this process the interpreter performs three steps:

1. Ship off the code for execution.
2. Process the statements of your script in a sequential fashion.
3. Compile the source code to an intermediate format known as bytecode.

Identify the correct order of these steps:

1 → 3 → 2

2 → 3 → 1 ✓

1 → 2 → 3

3 → 2 → 1

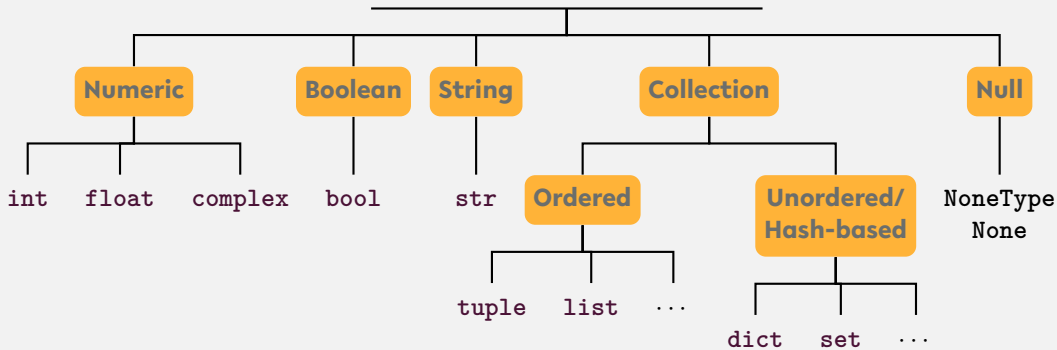
**Programming
Basics**

**Data Types &
Mutability**

**Evaluation Or-
der**

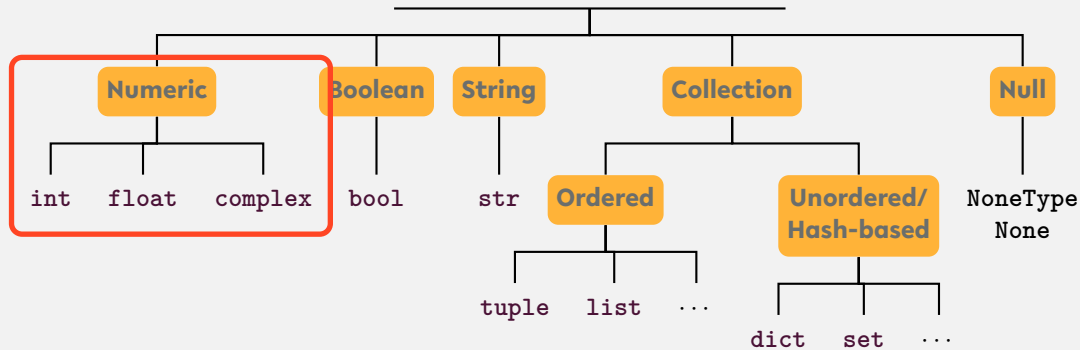
**Conditions &
Comparisons**

Python Data Types



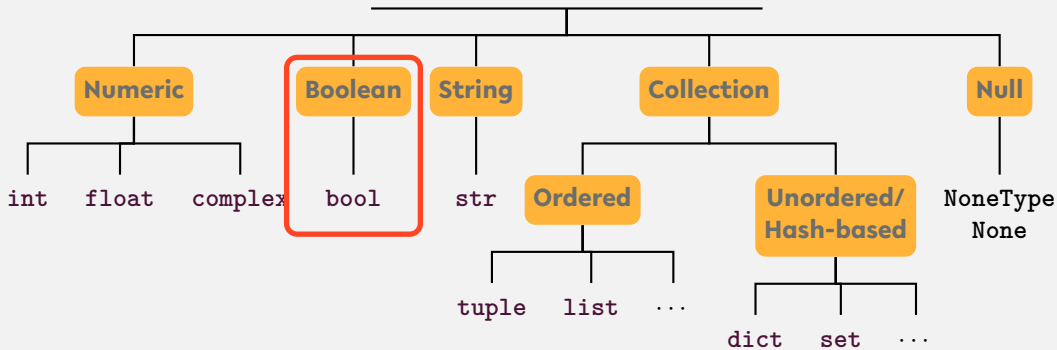
... and user-defined types

Python Data Types



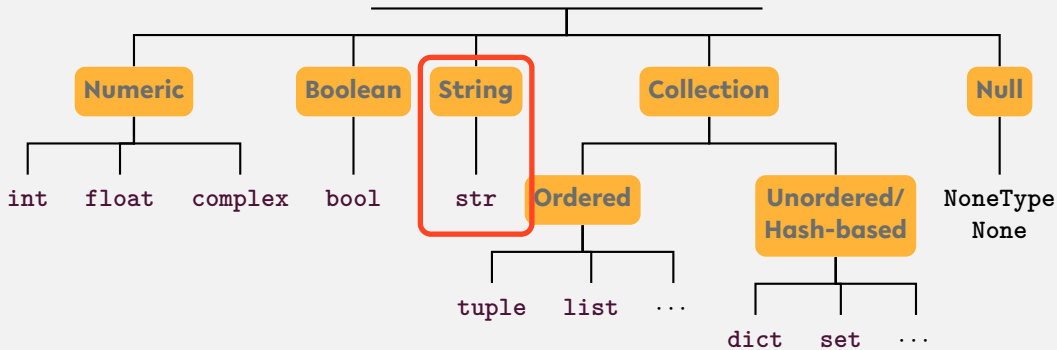
... and user-defined types

Python Data Types



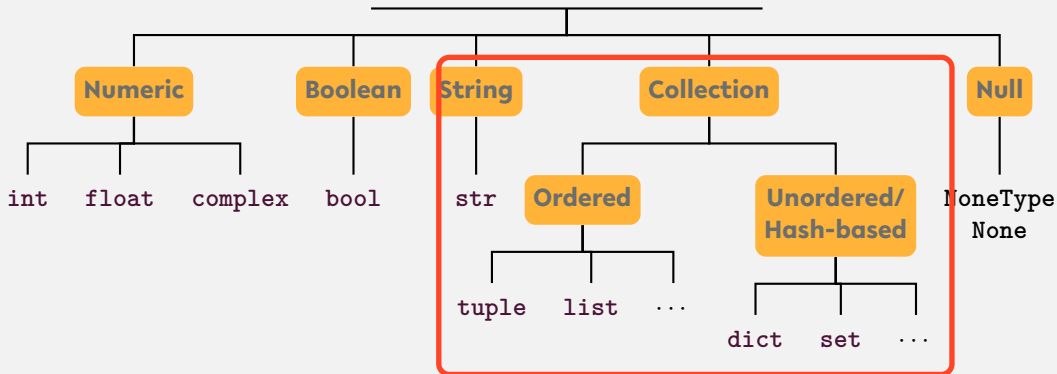
... and user-defined types

Python Data Types



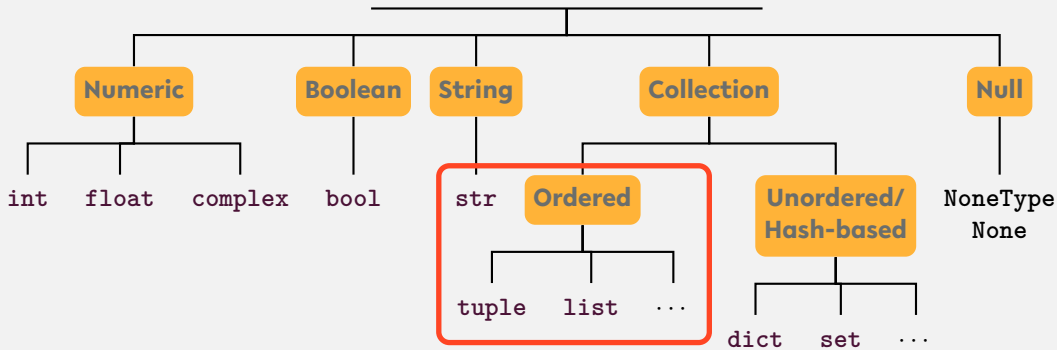
... and user-defined types

Python Data Types



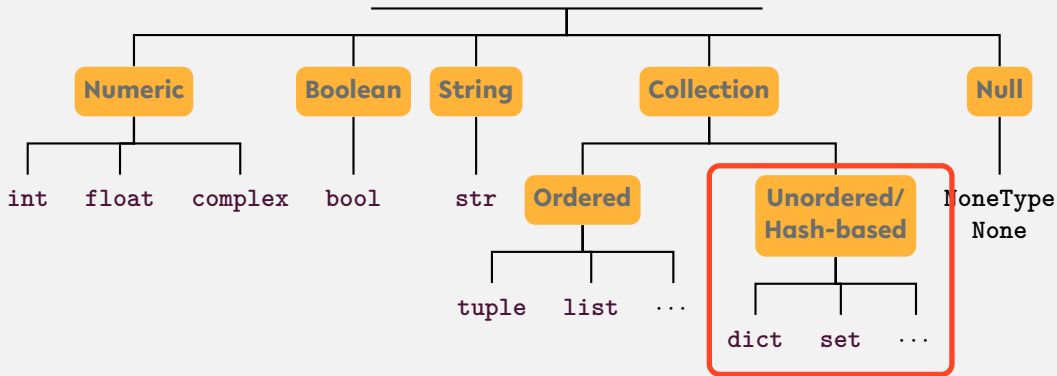
... and user-defined types

Python Data Types



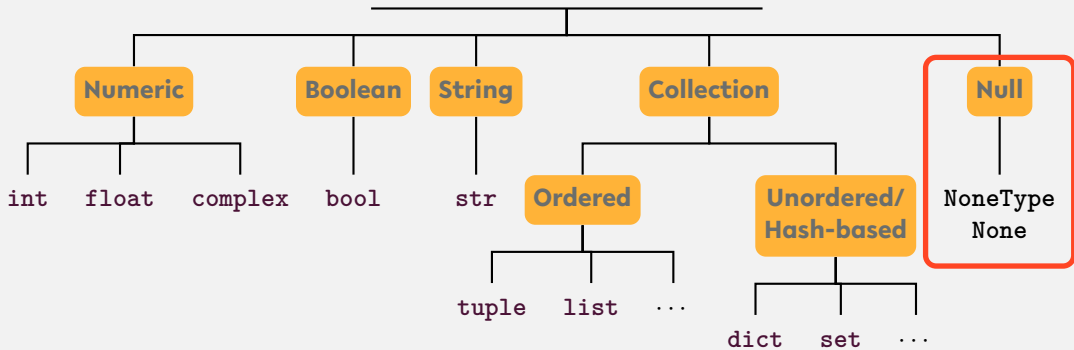
... and user-defined types

Python Data Types



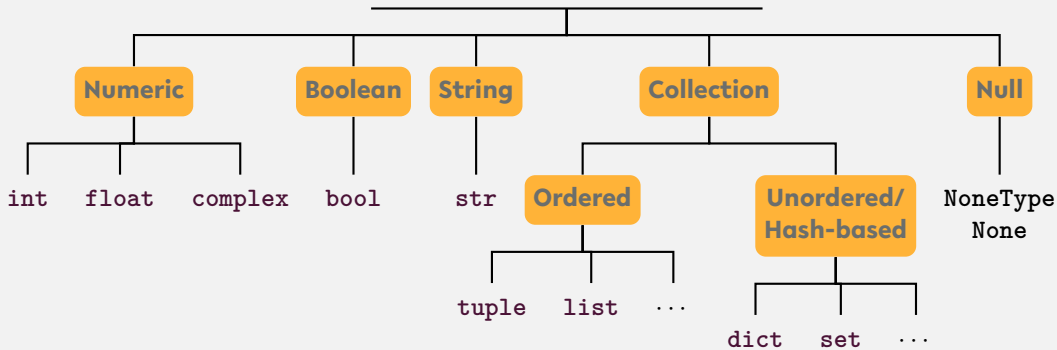
... and user-defined types

Python Data Types



... and user-defined types

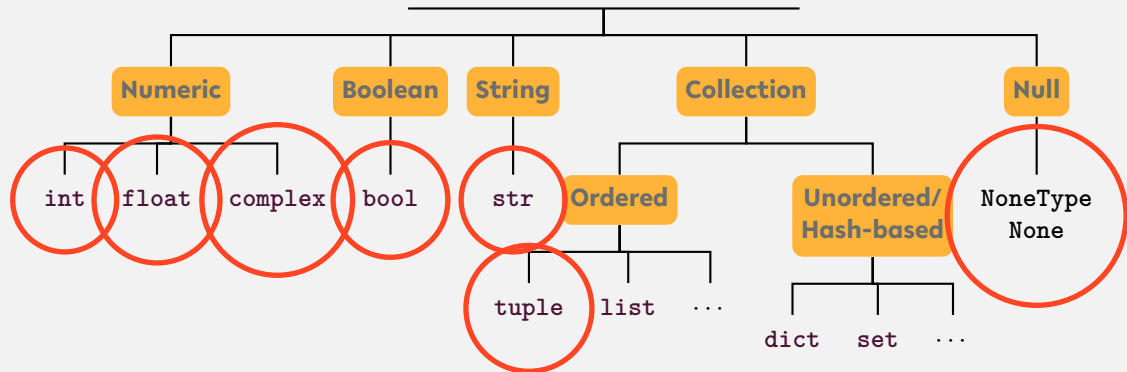
Python Data Types



... and user-defined types

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

Python Data Types



... and user-defined types

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

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`.

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`.
- ❖ *Fun fact: CPython binds integers from -5 to 256 on startup.*

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

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

**Programming
Basics**

**Data Types &
Mutability**

**Evaluation Or-
der**

**Conditions &
Comparisons**

Operator precedence

Parentheses (...)

Exponents **

Multiplication and Division * / // %

Addition and Substraction + -

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

Expression evaluation

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

$$\begin{array}{c} (5 - 1) * ((7 + 1) / (3 - 1)) \\ \downarrow \\ 4 * ((7 + 1) / (3 - 1)) \\ \downarrow \\ 4 * ((8) / (3 - 1)) \\ \downarrow \\ 4 * (8 / 2) \\ \downarrow \\ 4 * 4.0 \\ \downarrow \\ 16.0 \end{array}$$

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

**Programming
Basics**

**Data Types &
Mutability**

**Evaluation Or-
der**

**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')
```

Boolean operators and comparisons

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

Variables		Boolean expression		
a	b	<code>not a</code>	<code>a and b</code>	<code>a or b</code>
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 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

```
1 a = list()
2 b = 1
3 if b not in a:
4     print('b is contained in collection a')
5 else:
6     print('b is not contained in collection a')
```

Recap

Summary

- ❖ Computer architecture and Python compiler
- ❖ Python data types: `int`, `float`, `str`, `tuple`, `list`, `dict`, ...
- ❖ Operator precedence
- ❖ `if/else` clause
- ❖ Comparators: `==`, `!=`, `>`, `<`, `is`, `in`, ...

What comes next?

- ❖ Familiarize yourself with Spyder
- ❖ Write your first program!
- ❖ Due date for this week's exercises is **Saturday, May 2, 2020.**

Next lecture: Programming & Python basics continued ...

Spyder

The screenshot shows the Spyder Python IDE interface. The main editor window displays a file named `temp.py` with the following content:

```
1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4 This is a temporary script file.
5 """
6
7
8
```

The right-hand side of the interface contains a **Console** panel with a **Usage** tooltip. The tooltip text reads:

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](#)

Below the tooltip, the **Console 1/A** panel shows the following output:

```
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]:
```

The status bar at the bottom indicates: `Kite: ready`, `conda: base (Python 3.7.6)`, `Line 8, Col 1`, `UTF-8`, `LF`, `RW`, and `Mem 67%`.