

Applied Data Analytics

Basic Python

If conditions

Hans-Martin von Gaudecker and Aapo Stenhammar

Contents

- `if`, `elif`, and `else`
- More on Booleans
- More complex conditions

Motivation

- So far, all of our instructions in Python were very explicit
- There was no way of reacting to different situations:
 - Collecting elements that fulfil a condition
 - Doing different things singles and couples
 - ...
- This is what if conditions are for

Example: clipping a number

```
>>> number = -3.1

>>> if number < -3:
...     clipped = -3.0
... elif number > 3:
...     clipped = 3.0
... else:
...     clipped = number

>>> clipped
-3.0
```

- **if**, **elif**, and **else** are special keywords
- End each condition with a **:**
- What happens if that condition is **True** needs to be indented by 4 spaces and can span one or multiple lines
- Code following **False** conditions is skipped
- **elif x:** is the same as **else:** + nested **if x:**

More on Booleans

```
>>> bool(0)  
False
```

```
>>> bool(-1)  
True
```

```
>>> bool(1)  
True
```

```
>>> bool([])  
False
```

```
>>> bool([1, 2, 3])  
True
```

```
>>> bool("")  
False
```

```
>>> bool("abc")  
True
```

- What is not a Boolean can be converted to a Boolean
- This conversion happens implicitly after **if** and **elif**
- Can be useful and elegant but might compromise readability
- Rules of thumb:
 - 0 is **False**-ish
 - Other numbers are **True**-ish
 - Len-0 collections are **False**-ish
 - Len>0 collections are **True**-ish

More complex conditions

- Remember operators from "Assignments and Scalar Types":
 - **and**
 - **or** (inclusive)
 - **not**
- Example:

```
if a > b and b > some_cutoff:  
    do_something()  
else:  
    do_something_else()
```