J. Hageman
Hanze *University of Applied Sciences*
Groningen

Bioinformatics data processing and –analysis using the scripting language Python

Introduction to programming with Python

Booleans

Booleans

There are only two possible Booleans:

- True
- False

- In Python you can convert many datatypes to Boolean with the bool() function
- For every if/elif/else statement, expressions will be converted to Booleans

Booleans

```
>>> True
True
>>> False
False
>>> num = 4
>>> if num == 4: #evaluates as True
        print('num is 4')


num is 4
>>> num == 4
True
>>>
```

This should be easy to follow by now

Booleans

But many cases will be more complex. Example:

```
>>> list1 = []
>>> if not list1:
        list1.append("test_1")
else:
        list1.append("test_2")
```

This expression will test if the list is empty. Empty lists are evaluated as False!

```
>>> list1
['test_1']
```

Booleans

But many cases will be more complex. Example:

```
>>> list1
['test_1']
>>> if not list1:
        list1.append("test_1")
else:
        list1.append("test_2")


>>> list1
['test_1', 'test_2']
>>>
```

Now the list was not empty so the else clause will be executed.

Booleans

To understand this code, it is important that you understand how Booleans work.

```
>>> if not list1 == []:
        list1.append("test_1")
else:
        list1.append("test_2")


>>>
```

Probably this is more easy for you to understand but Python programmers do often directly evaluate variables for Bool types!

Booleans

For every if, elif, else statement, an expression is converted to a bool:

```
>>> bool([])
False
>>> bool(["test1"])
True
>>> bool([""])
True
>>>
```

Empty lists are False. As soon as a list contains an element (even an empty string) it is True

True/False in Python

So what is True and what is False?
This might seem complicated but it is not:
False is:

- None
- False
- 0 (zero as integer or float)
- empty sequence such as "" [] ()
- empty dictionary or set {}

Consider the rest True! (some details left out for simplicity)

Boolean Operators

There are three Boolean operations:

- or
- and
- not

They have a priority order: **or** goes first, than **and** and then **not**

To understand this:
See the next slides

```
>>> 1 and 2 or 3
2
>>> 1 and 2 and 3
3
>>> 1 or 2 or 3
1
>>>
```

Operators

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
>>>
```

As soon as an **and** expression finds False: it's False.

As soon as an **or** expression finds True: it's True.

Boolean Operator Priority

- **or** is a short-circuit operator, so it only evaluates the second argument if the first one is **false**.

bool(0) is False so Python will continue with bool(1)

bool(1) is True so Python will stop. No need to continue.

```
>>> 0 or 1
1
>>> 1 or 2
1
>>> False or False or False or False or True or False
True
>>> |
```

Python will stop here. No need to continue.

Boolean Operator Priority

- **and** is also short-circuit operator, so it only evaluates the second argument if the first one is true.

bool(0) is False so Python will stop here.

bool(1) is True so Python will continue.

```
>>> 0 and 1
0
>>> 1 and 2
2
>>> True and True and True and True and True and False and True
False
>>> 
```

Python will stop here. No need to continue.

Short circuit operator

```
>>> def return_true():
        print("running return_true")
        return True

>>> def return_false():
        print("running return_false")
        return False

>>> return_true() or return_false()
running return_true
True
>>> return_true() and return_false()
running return_true
running return_false
False
>>> return_false() or return_true()
running return_false
running return_true
True
>>> return_false() and return_true()
running return_false
False
>>>
```

Two functions are defined: One returns True, the other False

return_false() does not run because the **or** statement has received True already

return_true() does not run because the **and** statement has received False already

Brain heater

```
1 and not (”Yes” == “Yes” or False) and not [] or 5 != 6
```

True or False?

Brain heater

```
1 and not (”Yes” == “Yes” or False) and not [] or 5 != 6
```

Not so hard as it seems.
First solve any equality checks:

```
1 and not (True or False) and not [] or True
```

Find each and/or in parenthesis ():

```
1 and not (True) and not [] or True
```

Find each not and invert it:

```
1 and False and True or True
```

Find any remaining and/or and solve them:

```
True and False and True or True
```

```
True
```

Brain heater

```
>>> 1 and not ('Yes' == 'Yes' or False) and not [] or 5 != 6
True
>>> 
```