# Python Workshop

**Toni Gossmann**

University of Sheffield

May 2017

# Today's plan

Brief introduction to Python

First steps (interactive)

- Primitive data types
- Variables and collections

Control flow and modules

Parsing and manipulating a text file

Pipelines and external programs

# The other Python instructors (in alphabetical order)

- Henry Barton
- Mathias Bockwoldt
- Pádraic Corcoran
- Joseph Palmer

# Setup your account

- Follow the instructions on the webpage:
  `https://github.com/tonig-evo/tutorial_python`
- Login to Iceberg (click LATER for Java update and RUN)
- Installation time: 5-10 min

# Outline

# What is Python

- Programming language, developed by Guido van Rossum in 1991
- Interpreted higher-level language
- Interactive use or as scripts (*.py)
- Uses whitespace indentation to delimit code blocks (rather than curly brackets or keywords)

# Philosophy

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.

# Today's content: general session

- Primitive data types and operators (Interactive)
- Variables and collections (Interactive)
- Scripts and control flow (loops)
- Modules in Python
- Parsing and manipulating text files (Henry)
- Pipelines and external programs (Henry)

# Today's content: optional sessions

- Numpy and scipy (math and matrices)
- Biopython (Bio data files)
- Plotting (seaborn and Rpy2)
- Egglib (Population genetic package)

# Outline

# Setup your account

- Go to:
  `https://github.com/tonig-evo/tutorial_python`
- Open **python_pres.pdf** and click on the **Download** button
- Open another Interactive Job on Iceberg
- **Type python in your shell and press enter!**

# Primitive datatypes: Numbers

```
####################################################
## 1. Primitive Datatypes and Operators
####################################################

# You have numbers
3  # => 3

# Math is what you would expect
1 + 1   # => 2
8 - 1   # => 7
10 * 2   # => 20
35 / 5   # => 7
```

# Simple calculations

```
# Division is a bit tricky. It is integer division and
    ↪ floors the results
# automatically in Python 2.x. In Python 3 "/" is a "
    ↪ real" division.
5 / 2   # => 2

# To fix division we need to learn about floats.
2.0     # This is a float
11.0 / 4.0   # => 2.75 ahhh...much better

# Truncation or Integer division
5 // 3      # => 1
5.0 // 3.0 # => 1.0 # works on floats too

# Modulo operation
7 % 3 # => 1

# Enforce precedence with parentheses
(1 + 3) * 2   # => 8
```

# Boolean variables

```
# Boolean values are primitives (remember to
    ↪ capitalise!)
True
False

# negate with not
not True   # => False
not False   # => True

# Equality is ==
1 == 1   # => True
2 == 1   # => False

# Inequality is !=
1 != 1   # => False
2 != 1   # => True
```

# Comparisons

```
# More comparisons
1 < 10   # => True
1 > 10   # => False
2 <= 2   # => True
2 >= 2   # => True

# Comparisons can be chained!
1 < 2 < 3   # => True
2 < 3 < 2   # => False
```

# Strings

```
# Strings are created with " or '
"This is a string."
'This is also a string.'

# Strings can be added too!
"Hello " + "world!"  # => "Hello world!"

# A string can be treated like a list of characters
"This is a string"[0]  # => 'T'

# % can be used to format strings, like this:
"%s can be %s" % ("strings", "interpolated")

"{0} can be {1}".format("strings", "formatted")

"{name} wants to eat {food}".format(name="Bob", food="
    ↪ lasagna")
```

# None object

```
# None is an object
None   # => None

# Don't use the equality "==" symbol to compare
    ↪ objects to None.
# "None == anything" will always yield False
# Use "is" instead
"etc" is None   # => False
None is None   # => True
None == None   # => False

# The 'is' operator tests for object identity. This
    ↪ isn't
# very useful when dealing with primitive values, but
    ↪ is
# very useful when dealing with objects.
```

# Printing

```
##############################################
## 2. Variables and Collections
##############################################

# Python has a print function, available in versions
    ↪ 2.7 and 3...
print("I'm␣Python␣2.7␣or␣3.␣Nice␣to␣meet␣you!")
# and an older print statement, in all 2.x versions
    ↪ but removed from 3.
print "I'm␣also␣Python,␣but␣only␣version␣2.x!"
```

# Declare a variable

```
# No need to declare variables before assigning to
    ↪ them.
some_var = 5     # Convention is to use
    ↪ lower_case_with_underscores
some_var  # => 5

# Accessing a previously unassigned variable is an
    ↪ exception.
# See Control Flow to learn more about exception
    ↪ handling.
some_other_var  # Raises a NameError
```

# List - a simple collector

```
# Lists store sequences
li = []
# You can start with a prefilled list
other_li = [4, 5, 6]

# Add stuff to the end of a list with append
li.append(1)     # li is now [1]
li.append(2)     # li is now [1, 2]
li.append(4)     # li is now [1, 2, 4]
li.append(3)     # li is now [1, 2, 4, 3]
# Remove from the end with pop
li.pop()         # => 3 and li is now [1, 2, 4]
# Let's put it back
li.append(3)     # li is now [1, 2, 4, 3] again.
```

# Access a list

```
# Access a list like you would any array
li[0]   # => 1
# Look at the last element
li[-1]   # => 3

# Looking out of bounds is an IndexError
li[4]   # Raises an IndexError

# You can look at ranges with slice syntax.
# (It's a closed/open range for you mathy types.)
li[1:3]   # => [2, 4]
# Omit the beginning
li[1:]   # => [2, 4, 3]
li[2:]   # => [4, 3]
# Omit the end
li[:3]   # => [1, 2, 4]
li[:-1]   # => [1, 2, 4]
```

# Access a list - more fancy

```
# Select every second entry
li[::2]    # =>[1, 4]
# Revert the list
li[::-1]    # => [3, 4, 2, 1]
# Use any combination of these to make advanced slices
# li[start:end:step]

# Remove arbitrary elements from a list with "del"
del li[2]    # li is now [1, 2, 3]

# You can add lists
li + other_li    # => [1, 2, 3, 4, 5, 6] - Note: values
    ↪  for li and for other_li are not modified.

# Concatenate lists with "extend()"
li.extend(other_li)    # Now li is [1, 2, 3, 4, 5, 6]
```

# Unpack a list into variables

```
# You can unpack lists into variables
a, b, c = [1, 2, 3]      # a is now 1, b is now 2 and c
    ↪  is now 3
# Now look how easy it is to swap two values
e, d = d, e      # d is now 5 and e is now 4
# The number of elements to unpack must fit
a, b, c = [1, 2, 3, 4]    # - raises a ValueError
```

# Dictionary

```
# Dictionaries store mappings
empty_dict = {}
# Here is a prefilled dictionary
filled_dict = {"one": 1, "two": 2, "three": 3}

# Look up values with []
filled_dict["one"]   # => 1

# Get all keys as a list with "keys()"
filled_dict.keys()   # => ["three", "two", "one"]
# Note - Dictionary key ordering is not guaranteed.
# Your results might not match this exactly.

# Get all values as a list with "values()"
filled_dict.values()   # => [3, 2, 1]
# Note - Same as above regarding key ordering.
```

# Dictionary

```
# Check for existence of keys in a dictionary with "in
    ↪ "
"one" in filled_dict    # => True
1 in filled_dict    # => False

# Looking up a non-existing key is a KeyError
filled_dict["four"]    # KeyError

# Use "get()" method to avoid the KeyError
filled_dict.get("one")    # => 1
filled_dict.get("four")    # => None
```

# Sets

```
# Initialize a "set()" with a bunch of values
some_set = set([1, 2, 2, 3, 4])   # some_set is now
    ↪ set([1, 2, 3, 4])

# Since Python 2.7, {} can be used to declare a set
filled_set = {1, 2, 2, 3, 4}   # => {1, 2, 3, 4}

# Do set intersection with &
other_set = {3, 4, 5, 6}
filled_set & other_set   # => {3, 4, 5}

# Do set union with |
filled_set | other_set   # => {1, 2, 3, 4, 5, 6}

# Do set difference with -
{1, 2, 3, 4} - {2, 3, 5}   # => {1, 4}
```

# Excercise

- `https://github.com/tonig-evo/tutorial_python`
- Questions 1a and 1b

# Outline

# Create your own script on Iceberg

- Create: > my_first_script.py
- Edit: gedit my_first_script.py
- Run: python my_first_script.py

# If statement

```
####################################################
## 3. Control Flow
####################################################

# Let's just make a variable
some_var = 5

# Here is an if statement. Indentation is significant
    ↪ in python!
# prints "some_var is less than 10"
if some_var > 10:
    print("some_var is totally bigger than 10.")
elif some_var < 10:    # This elif clause is optional.
    print("some_var is less than 10.")
else:              # This is optional too.
    print("some_var is indeed 10.")
```

# Loop through a list of strings

```
"""
For loops iterate over lists
prints:
    dog is a mammal
    cat is a mammal
    mouse is a mammal
"""
for animal in ["dog", "cat", "mouse"]:
    # You can use {} and format() to interpolate
        ↪ formatted strings
    print("{0} is a mammal".format(animal))
```

# Loop through a range of integers

```
"""
"range(number)" returns a list of numbers in Python 2.
    ↪ x. In Python 3, range(number) returns
a special generator. To turn it into a list, use list(
    ↪ range(number))
The list will contain numbers from zero to one less
    ↪ than the given number
prints:
    0
    1
    2
    3
"""
for i in range(4):
    print(i)
```

# Loop through a sliced range of integers

```
"""
As slicing, range may take start, end, step.
prints:
    3
    5
    7
"""
for i in range(3, 9, 2):
    print(i)
```

# Import modules

```
####################################################
## 5. Modules
####################################################

# You can import modules
import math
print(math.sqrt(16))  # => 4

# You can get specific functions from a module
from math import ceil, floor
print(ceil(3.7))  # => 4.0
print(floor(3.7))   # => 3.0
```

# Import modules

```
# You can shorten module names
import math as m
math.sqrt(16) == m.sqrt(16)    # => True

# Python modules are just ordinary python files. You
# can write your own, and import them. The name of the
# module is the same as the name of the file.

# You can find out which functions and attributes
# defines a module.
import math
dir(math)

# You can find out which modules are available
help("modules")
```

# Excercise

- `https://github.com/tonig-evo/tutorial_python`
- Questions 2 and 3

# Outline

# Outline