Introductory Linux Course

Python I

Martin Dahlö – UPPMAX

Author: Nina Fischer
Dept. for Cell and Molecular Biology, Uppsala University

August, 2018
Outline

- Python basics – get started with Python
- Data types
- Control structures
  - Loops: for and while
  - Conditions: if-else
- Our first Python program
Python Basics

- We use Python 3 (version 3.6.0) in this course. For this we need to load a module:

  ```bash
  [ninaf@rackham1 ~]$ module load python3/3.6.0
  ```

- When you type `python3` on the command line the Python-Interpreter comes ready:

  ```bash
  [ninaf@rackham1 ~]$ python3
  Python 3.6.0 (default, Nov 21 2017, 09:52:46)
  [GCC 4.4.7 20120313 (Red Hat 4.4.7-18)] on linux
  Type "help", "copyright", "credits" or "license" for more information.
  >>>
  ```

  **Note:** If you type "python", you start Python 2 (version 2.7.5)!
Python Basics

- With `quit()`, `exit()`, or `Ctrl-D` you can close the Python-interpreter

```
[ninaf@rackham1 ~]$ python3
Python 3.6.0 (default, Aug 4 2017, 00:39:18)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-16)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
[ninaf@rackham1 ~]$ 
```

- With arrow keys "up" (↑) and "down" (↓) you can scroll through previous commands
Python as Calculator

```python
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 12.45 / 100 + 7.5e-3
0.132
>>> 
```
Python Output

- By default the output appears in the terminal (the window you are working in)
- With the `print()` function you can write text

```python
>>> print("Welcome to our Introductory Linux Course!")
Welcome to our Introductory Linux Course!
>>> print(2)
2
>>> print(2+2)
4
```
Python Assignment

- We can store text or values in variables (assign text or values to variables) in order to
  - Conveniently refer to them
  - Separate Python code from data

```python
>>> greeting = "Welcome to our Introductory Linux Course!"
>>> print(greeting)
Welcome to our Introductory Linux Course!
>>> number = 2
>>> print(number)
2
```

- The assignment is done with the equal sign (=)
Names of variables may be chosen freely, but

- must consist of a single word (no blanks)
- must not contain special characters except "_"
- must not begin with a number

Valid names are: my_variable, Value15

Invalid names are: my-variable, 15th_value
>>> greeting = "Welcome to our Introductory Linux Course!"
>>> print(greeting)
Welcome to our Introductory Linux Course!
>>>
>>> greeting = "Welcome to our Introductory Linux Course!"
>>> print(greeting)
Welcome to our Introductory Linux Course!

>>> number = 2
>>> print(number)
2

>>>
>>> greeting = "Welcome to our Introductory Linux Course!"
>>> print(greeting)
Welcome to our Introductory Linux Course!

>>> number = 2
>>> print(number)
2

>>> print(greeting*number)
Welcome to our Introductory Linux Course!Welcome to our Introductory Linux Course!

>>>
>>> greeting = "Welcome to our Introductory Linux Course!"
>>> print(greeting)
Welcome to our Introductory Linux Course!

>>> number = 2
>>> print(number)
2

>>> print(greeting*number)
Welcome to our Introductory Linux Course!Welcome to our Introductory Linux Course!

>>> print(greeting*4)
Welcome to our Introductory Linux Course!
Welcome to our Introductory Linux Course!
Welcome to our Introductory Linux Course!
Welcome to our Introductory Linux Course!
>>> greeting = "Welcome to our Introductory Linux Course!"
>>> print(greeting)
Welcome to our Introductory Linux Course!
>>> 
>>> number = 2
>>> print(number)
2
>>> 
>>> print(greeting*number)
Welcome to our Introductory Linux Course!Welcome to our Introductory Linux Course!
>>> 
>>> greeting = "Hello!\n"
>>> print(greeting*4)
Hello!
Hello!
Hello!
Hello!

>>>
Python knows different *types* of data

- **int/float** for numeric data (integer or floating point numbers)
- **str** for text (or sequences of characters, so-called strings)
Python knows different **types** of data

- **int/float** for numeric data (integer or floating point numbers)
- **str** for text (or sequences of characters, so-called strings)
Data Types

>>> 4 + 2
6
>>>
Data Types

>>> 4 + 2
6
>>> 4 + 2.0
6.0
>>>
Data Types

>>> 4 + 2
6
>>> 4 + 2.0
6.0
>>> 8 / 5
1.6

Note: In Python 2 division of two integer numbers results in an integer: 8/5 = 1
Data Types

>>> 4 + 2
6
>>> 4 + 2.0
6.0
>>> 8 / 5
1.6
>>> 8 / 5.0
1.6
>>> 

**Note:** In Python 2 division of two integer numbers results in an integer: 8/5 = 1
The function \texttt{type()} yields information on the type of a variable
The function `type()` yields information on the type of a variable.
Data Types

- The function `type()` yields information on the type of a variable.
Python Strings

- In Python operations with strings of characters can be done in a very convenient way.

- Given are two strings $A = \text{"ABCD"}$ and $B = \text{"EFG"}$.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Python</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection of a character</td>
<td>$A[0], \ B[2]$</td>
<td>&quot;A&quot;, &quot;G&quot;</td>
</tr>
<tr>
<td>Concatenation</td>
<td>$B + A$</td>
<td>&quot;EFGABCD&quot;</td>
</tr>
<tr>
<td>Substring</td>
<td>$A[1:3]$</td>
<td>&quot;BC&quot;</td>
</tr>
<tr>
<td>Prefix</td>
<td>$A[:2]$</td>
<td>&quot;AB&quot;</td>
</tr>
<tr>
<td>Suffix</td>
<td>$A[1:]$</td>
<td>&quot;BCD&quot;</td>
</tr>
</tbody>
</table>

Note: Python starts counting from zero!
Methods

```python
>>> greeting = "Welcome!"
>>> dir(greeting)
```
Methods

```python
>>> greeting = "Welcome!"

```
Methods

- The function `dir()` yields information on the methods of a specific data type or object.

```python
>>> greeting = "Welcome!"
>>> dir(greeting)
[...,'count',...]
>>> 
```
The function `help()` yields further information on how to use these methods (press "q" to quit)
Methods

```python
>>> greeting = "Welcome!"
>>> dir(greeting)
[...,'count',...]

>>> help(greeting.count)

Help on built-in function count:

count(...)
S.count(sub[, start[, end]]) -> int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

>>> greeting.count('e')
2
```
Methods

```python
>>> greeting = "Welcome!"
>>> dir(greeting)
[...,'count',...]
>>> help(greeting.count)
Help on built-in function count:

```
count(...)
S.count(sub[, start[, end]]) -> int
```

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

>>> greeting.count('e')
2
>>> greeting.count('e', 2)
1
```
Methods

```python
>>> greeting = "Welcome!"
>>> dir(greeting)
[...,'count',...]

>>> help(greeting.count)

Help on built-in function count:

count(...)  
S.count(sub[, start[, end]]) -> int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

>>> greeting.count('e')
2
>>> greeting.count('e', 2)
1
>>> greeting.count('e', 2, 4)
0
>>>```
Methods

```python
>>> greeting = "Welcome!"
>>> dir(greeting)
[...,'count',...]

>>> help(greeting.count)

Help on built-in function count:

``count``(...)

S.count(sub[, start[, end]]) -> int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

>>> greeting.count('e')
2
>>> greeting.count('e', 2)
1
>>> greeting.count('e', 2, 4)
0
>>> greeting.count('ome')
1
```
Python Lists

- **Lists** contain series of arbitrary values (to be precise: objects)
- Lists are defined by writing the individual values separated by commas inside square brackets
- Lists can contain different data types

```python
>>> l = [1, 2, "ABC", [3, "DEF"]]
```

- One can define empty lists:

```python
>>> l = []
```
Python Lists

- With `append` one can add elements to a list
- Lists can be concatenated with the operator `+`
- Lists can be accessed via index operations in the same way as strings

```python
>>> l = []
>>> l.append(1)
>>> l.append(2)
>>> print(l)
[1, 2]
```
With **append** one can add elements to a list

- Lists can be concatenated with the operator `+`
- Lists can be accessed via index operations in the same way as strings

```python
>>> l = []
>>> l.append(1)
>>> l.append(2)
>>> print(l)
[1, 2]
>>> m = [3, 4, 5]
>>> n = l + m
>>> print(n)
[1, 2, 3, 4, 5]
```
Python Lists

- With `append` one can add elements to a list
- Lists can be concatenated with the operator `+`
- Lists can be accessed via index operations in the same way as strings

```python
>>> l = []
>>> l.append(1)
>>> l.append(2)
>>> print(l)
[1, 2]
>>> m = [3, 4, 5]
>>> n = l + m
>>> print(n)
[1, 2, 3, 4, 5]
>>> print(n[2])
3
```
Python Lists

- Python has more useful built-in functions

```python
>>> l = [170, 50, 3, 244]
>>> print(min(l))
3
>>> 
```
Python Lists

- Python has more useful built-in functions

```python
>>> l = [170, 50, 3, 244]
>>> print(min(l))
3
>>> print(max(l))
244
```
Python Lists

- Python has more useful built-in functions

```python
>>> l = [170, 50, 3, 244]
>>> print(min(l))
3
>>> print(max(l))
244
>>> print(sorted(l))
[3, 50, 170, 244]
```
Python Lists

- Python has more useful built-in functions

```python
>>> l = [170, 50, 3, 244]
>>> print(min(l))
3
>>> print(max(l))
244
>>> print(sorted(l))
[3, 50, 170, 244]
>>> print(sum(l))
467
```
Python Lists

- Python has more useful built-in functions

```python
>>> l = [170, 50, 3, 244]
>>> print(min(l))
3
>>> print(max(l))
244
>>> print(sorted(l))
[3, 50, 170, 244]
>>> print(sum(l))
467
>>> print(len(l))
4
```
Dictionaries (dict) uses keys instead of index numbers, which makes them unordered.

Dicts are defined by writing key-value pairs separated by commas inside curly brackets.

Dicts can contain different data types.

```
>>> D = {"key1":5, "nextKey":"look, a string", "l":[1,2,3]}
```

One can define empty dicts:

```
>>> l = {}
```
You add elements by assigning a value to a key

```python
>>> D = {}
>>> D['new key'] = "ABC"
>>> D[2] = 42
>>> D['5'] = 99
>>> 
```
You add elements by assigning a value to a key

```python
>>> D = {}
>>> D['new key'] = "ABC"
>>> D[2] = 42
>>> D['5'] = 99
>>> print(D)
{'5': 99, 2: 42, 'new key': 'ABC'}
```
You add elements by assigning a value to a key
Python Dictionary

- You add elements by assigning a value to a key.

```python
>>> D = {}
>>> D['new key'] = "ABC"
>>> D[2] = 42
>>> D['5'] = 99
>>> print(D)
{'5': 99, 2: 42, 'new key': 'ABC'}

>>> print(D['5'] + D[2])
141

>>> D['1'] = [19, 83, 1, 15]
>>> print(D['1'][3])
15
```
Control Structures

- Control structures determine the logical flow of a program
- There are two types of key control structures in Python:
  - Loops: `for`, `while`
  - Conditions: `if-else`
- These two types of control structures permit the modeling of all possible program flows
Python Loops

- One can iterate with for-loops over elements of a list (e.g., list, string)
- The head defines the loop variable
- The body is executed for each of the values of the loop variable

```python
>>> l = [170, 50, 3, 244]  # list
>>> for i in l:
...     print(i)    # head with loop variable i
...     print(i)    # body
...  # press ENTER a second time
```
Python Loops

- One can iterate with for-loops over elements of a list (e.g., list, string)
- The head defines the loop variable
- The body is executed for each of the values of the loop variable

```python
>>> l = [170, 50, 3, 244]  # list
>>> for i in l:
...     print(i)  # head with loop variable i
...  # body
...  # press ENTER a second time
170
50
3
244
>>> 
```
Python Loops

- One can iterate with for-loops over elements of a list (e.g., list, string)
- The head defines the loop variable
- The body is executed for each of the values of the loop variable

```python
>>> l = [170, 50, 3, 244]  # list
>>> for my_loop_variable in l:  # head with loop variable
    # body
    print(my_loop_variable)
```

```
170
50
3
244
```
Python Loops

- One can iterate with for-loops over elements of a list (e.g., list, string)
- The head defines the loop variable
- The body is executed for each of the values of the loop variable

```python
>>> l = [170, 50, 3, 244]  # list
>>> for i in l:
...    print(i)
...  # press ENTER a second time
```
Python Loops

- One can iterate with for-loops over elements of a list (e.g., list, string)
- The head defines the loop variable
- The body is executed for each of the values of the loop variable

```python
>>> l = [170, 50, 3, 244]  # list
>>> for i in l:
    ... print(i)  # head with loop variable i
    ... print(i)  # body
    ...  # press ENTER a second time

File "<stdin>", line 2
    print(i)  # press ENTER a second time
                      ^
IndentationError: expected an indented block
```
### Python Loops

- One can iterate with for-loops over elements of a list (e.g., list, string)
- The head defines the loop variable
- The body is executed for each of the values of the loop variable

```python
>>> A = "ABCD"
>>> for i in A:
...     print(i)
... # press ENTER a second time
```
Python Loops

- One can iterate with for-loops over elements of a list (e.g., list, string)
- The head defines the loop variable
- The body is executed for each of the values of the loop variable

```python
>>> A = "ABCD"
# string
>>> for i in A:
# head with loop variable i
...     print(i)
# body
...     # press ENTER a second time
A
B
C
D
>>> 
```
Python Loops

- The body may contain multiple lines of code

```python
>>> A = "ABCD"
>>> l = []
>>> for i in A:
...     print(i)
...     l.append(i)
...
A
B
C
D
>>> # string
# head with loop variable i
# body
# body
# press ENTER a second time
```
Python Loops

- The body may contain multiple lines of code

```python
>>> A = "ABCD"
>>> l = []
>>> for i in A:
...     print(i)
...     l.append(i)
...
A
B
C
D
>>> print(l)
['A', 'B', 'C', 'D']
>>> 
```
The build-in function `range()` creates a list of values

The list elements can be used to iterate through during a for-loop

```
>>> for i in range(5):
...     print(i)
...          
0
1
2
3
4
```
Python Loops

- The build-in function `range()` creates a list of values
- The list elements can be used to iterate through during a for-loop

```python
>>> A = "ABCD"
>>> print(len(A))
4
>>> for i in range(len(A)):
...     print(i)
...
0
1
2
3
```
The build-in function `range()` creates a list of values

The list elements can be used to iterate through during a for-loop
Nested Loops

- The body of the first loop (also called outer loop) includes the body of the second loop (called inner loop)

- Prints all words of length 3 of $\Sigma_{DNA}$:

```python
>>> A = "ACGT"
>>> for i in A:
...     for j in A:
...         for k in A:
...             print(i + j + k)

AAA
AAC
AAG
AAT
ACA
etc.
```
Python While-Loops

- While-loops iterate as long as a certain condition is met
- Before each iteration of a while-loop is executed, the condition in the head is tested, while it is still true, the body is executed

```python
>>> A = "ABCD"
>>> i = 0
>>> while i < len(A):
...    print(A[i])
...    i = i + 1
```

```python
>>> A = "ABCD"
>>> for i in range(len(A)):
...    print(A[i])
```
Python Conditions

- Conditions limit the execution of parts of the program

- In Python this construct is called if-else

```python
>>> A = "ACGT"
>>> B = "AAT"
>>> if len(A) < len(B):
...     print("Sequence A is smaller than B.")
```

- If the condition is fulfilled the block following if is executed
Python Conditions

- Conditions limit the execution of parts of the program
- In Python this construct is called if-else

```python
>>> A = "ACGT"
>>> B = "AAT"
>>> if len(A) < len(B):
...     print("Sequence A is smaller than B.")
... else:
...     print("Sequence A is greater or equal than B.")
```

- If the condition is fulfilled the block following if is executed, otherwise the block after else is executed
Conditions limit the execution of parts of the program

In Python this construct is called if-else

```python
>>> A = "ACGT"
>>> B = "AAT"
>>> if len(A) < len(B):
...     print("Sequence A is smaller than B.")
... else:
...     print("Sequence A is greater or equal than B.")
```

For comparisons different operators are used

```python
==   equal
!<   not equal
<    less
>    greater
<=   equal to or less
>=   greater or equal
```
The if-else condition can be extended with elif to consider multiple conditions (as many as necessary)

```python
>>> A = "ABCD"
>>> if len(A) <= 3:
...     print("Sequence A is smaller or equal than 3.")
... elif (len(A) > 3) and (len(A) < 5):
...     print("Sequence A is greater than 3 and smaller than 5.")
... elif len(A) == 5:
...     print("Sequence A is equal to 5.")
... else:
...     print("Sequence A is greater than 5.")
```
Assignment:
store a value in a variable

```python
>>> A = "ACGT"
>>> B = "AAT"
>>> number = 5
>>> sum = 10
```

Comparison:
compare two values

```python
>>> if len(A) == len(B):
...     print("Sequence", A, "is equal to sequence", B)
... else:
...     print("Sequence", A, "is not equal to sequence", B)
...
Sequence ACGT is not equal to sequence AAT
```
Writing and Executing a Program

- Python allows not only interactive work, but also the execution of a full program which is saved as a file.
- It reads the full program and executes each line consecutively, starting with the first.
- This program calculates the product of two numbers:

```python
x = 15
print("x = ", x)
y = 456
print("y = ", y)
print("Product of x and y is ", x*y)
```
One can use an arbitrary editor (gedit, nano, emacs, vi, ...) to write the program and save it as a file, named e.g. "product.py"

```
x = 15
print("x = ", x)
y = 456
print("y = ", y)
print("Product of x and y is", x\*y)
```
Writing and Executing a Program

- Python-Interpreter vs. editor written program

```python
>>> A = "ACGT"
>>> if len(A) <= 3:
...     print("Sequence A is smaller or equal than 3.")
>>> elif len(A) > 3 and len(A) < 5:
...     print("Sequence A is greater than 3 and smaller than 5.")
>>> elif len(A) == 5:
...     print("Sequence A is equal to 5.")
>>> else:
...     print("Sequence A is greater than 5.”)
```
Writing and Executing a Program

- To execute the program we call the interpreter from the command line using the filename as argument

[ninaf@rackham1 ~]$ python3 sequence.py
Sequence A is greater than 3 and smaller than 5.
References

- http://www.diveintopython.net
  A full book about Python freely available for download
- http://openbookproject.net/thinkcs/python/english2e/
  „How to think like a computer scientist“
  With examples in Python!

More information:

- On http://python.org
  For example:
  A Python Tutorial: https://docs.python.org/2/tutorial/
- https://www.codecademy.com/learn/learn-python
- You can easily install python on your own computer:
  http://www.python.org

  • Integrated development environment for python (IDLE)
    https://wiki.python.org/moin/IDLE or https://code.visualstudio.com/