Bioinformatical problem solving with Python

Wednesdays 17:30-19:00, M801

alexander.nater@uni-konstanz.de

List comprehension

- Get list of squared values from list:
  my_list = [1,2,3,4,5,6]

- squared_list = []
  for value in my_list:
      squared_list.append(value**2)

- squared_list = list(map(lambda x: x**2, my_list))

- squared_list = [x**2 for x in my_list]

List comprehension

- result = [expr for val in sequence if condition]

-
```
result = []
for val in sequence:
    if condition:
        result.append(expr)
```

- squared_list = [x**2 for x in my_list if x > 1]

Example

- Create a new list containing the result of adding the values in two list together for each element.

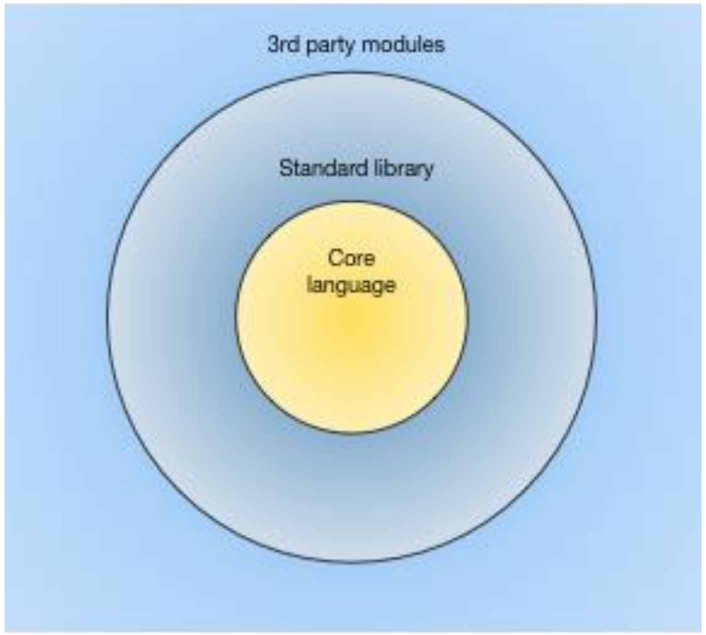
-
  ```
  la = [1,2,3,4,5,6,7,8,9,10]
  lb = list(reversed(la))
  new_list = [ x + y for x,y in zip(la, lb) ]
  ```

Exercise

- From some random text, obtain a list of capitalized words with more than four letters.

- string = “””Some random text … “””
  capitalized = [word.upper() for word in string.replace('.', '').split() if len(word) > 4]

Modules



- The Python core language has only very limited functionality.
- We can easily expand it by importing additional Python modules.

Importing modules

- import module_name

  import os
  os.getcwd()

- import module_name as new_name

  import os as opsys
  opsys.getcwd()

- from module_name import class_name

  from os import getcwd
  getcwd()

- from module_name import *

Running a script

- python3 scriptfile.py arg1 arg2 arg3 …

- Import the sys module to get access to command line arguments:

```
import sys
script_name = sys.argv[0]
arg1 = sys.argv[1]
arg2 = sys.argv[2]
arg3 = sys.argv[3]
…
```

Exercises

- Write a function that sums up a list of integers and save it in a Python file (*.py). Import your module from the interpreter and call the function.

- Write a script that takes multiple integers as command line arguments and prints the their sum.

Reading files

- Use the ‘open’ function with the ‘r’ argument to create a readable file object:
  my_file = open("infile.txt", 'r')
- File objects are normal Python objects:
  dir(my_file)
- File objects are iterable and can directly be used in for loops.
- File objects need to be closed after using them:
  my_file.close()

Reading files

- Common syntax to work with file objects:

```
with open("infile.txt", 'r') as my_file:
    do something with my_file …
```

- Ensures that file handles are always properly closed when the block ends.

Reading files

- The ‘read(n)’ method reads n bytes of the file into a string.
- The ‘readline’ method reads a line of the file into a string:
  ```
  line = my_file.readline()
  ```
- The ‘readlines’ method reads each line of the file into a list. → Requires lots of memory for large files!
  ```
  lines = my_file.readlines()
  ```
- The preferred way is to read large files line by line:
  ```
  for line in my_file:
      do something with line …
  ```

Writing to files

- Use the ‘open’ function with the ‘w’ (write) or ‘a’ (append) argument to create a writable file object:
  my_outfile = open("outfile.txt", 'w')
  my_outfile = open("outfile.txt", 'a')

- Use the ‘print’ function with the ‘file’ argument to write to the file:
  print(“Some text ”, var1, “ more text “, file=my_outfile)

- File objects need to be closed after using them:
  my_outfile.close()

Exercises

- Download the vcf file ‘Apo_cl2_scaffold2_5Mb.vcf’ from Github.
- The vcf file contains phased genotype data over 5 Mb on scaffold2 for 10 indiviudals (no missing data).
- Calculate mean heterozygosity for each individual and print the values together with the individual names to a text file.
- Get counts for each unique haplotype in the region scaffold2:2’000’000-2’020’000.