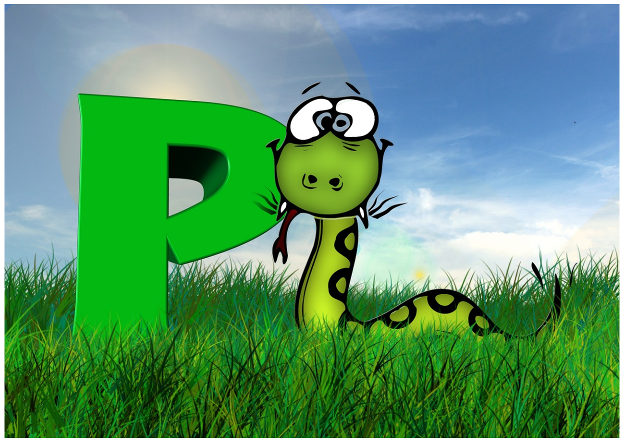
MODULES AND PACKAGES

PYTHON FOR GENOMIC DATA SCIENCE



What Are Modules?

- Modules in Python are simply Python files with the `.py` extension, which contain definitions of functions, or variables, usually related to a specific theme.

- Grouping related code into a module makes the code easier to understand and use.

Example

We could put all the functions we wrote to process a DNA sequence in a file called, for instance, `dnautil.py`

`dnautil.py`

```
#!/usr/bin/python
"""
dnautil module contains a few useful functions for dna sequence
"""
def gc(dna) :
     "this function computes the GC percentage of a dna sequence”
     nbases=dna.count('n')+dna.count('N')
     gcpercent=float(dna.count('c')+dna.count('C')+dna.count('g')
+dna.count('G'))/(len(dna)-nbases)
     return gcpercent

def has_stop_codon(dna,frame) :
   “This function checks if given dna sequence has in frame stop
codons.”
   stop_codon_found=False
        ...
```

Using Modules

Enter the Python interpreter and import the `dnautil.py` module with the following command:

```
>>> import dnautil
```

If you didn't put your `dnautil.py` file in your current directory, you might get an error:

```
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    import dnautil
ImportError: No module named 'dnautil'
```

Where Are The Modules?

When a module is imported, Python first searches for a built-in module with that name.

If a built-in module is not found, Python then searches for a file obtained by adding the extension `.py` to the name of the module that it's imported:

- in your current working directory,
- the directory where Python has been installed,
- in a path, i.e. a colon (':') separated list of file paths, stored in the environment variable PYTHONPATH.

The `sys.path` Variable

You can use the `sys.path` variable from the `sys` built-in module to check the list of all directories where Python looks for files:

```
>>> import sys
>>> sys.path
['', '/Users/mpertea/Documents', '/Library/Frameworks/
Python.framework/Versions/3.4/lib/python34.zip', '/
Library/Frameworks/Python.framework/Versions/3.4/lib/
python3.4', '/Library/Frameworks/Python.framework/
Versions/3.4/lib/python3.4/plat-darwin', '/Library/
Frameworks/Python.framework/Versions/3.4/lib/python3.4/
lib-dynload', '/Library/Frameworks/Python.framework/
Versions/3.4/lib/python3.4/site-packages']
```

Extending The Search Path

If the sys.path variable doesn't contain the directory where you put your module you can extend it:

```
>>> sys.path.append("/Users/mpertea/courses/python")
>>> sys.path
['', '/Users/mpertea/Documents', '/Library/Frameworks/
Python.framework/Versions/3.4/lib/python34.zip', '/
Library/Frameworks/Python.framework/Versions/3.4/lib/
python3.4', '/Library/Frameworks/Python.framework/
Versions/3.4/lib/python3.4/plat-darwin', '/Library/
Frameworks/Python.framework/Versions/3.4/lib/python3.4/
lib-dynload', '/Library/Frameworks/Python.framework/
Versions/3.4/lib/python3.4/site-packages', '/Users/
mpertea/courses/python']
```

Using Modules (cont'd)

```
>>> import dnautil
```

Now you can try using the functions in the `dnautil` module:

```
>>> dna="atgagggctaggt"
>>> gc(dna)
Traceback (most recent call last):
  File "<pyshell#38>", line 1, in <module>
    gc(dna)
NameError: name 'gc' is not defined
```

What happened?

You need to use the module name to access its functions:

```
>>> dnautil.gc(dna)
0.5384615384615384
```

Importing Names From A Module

You can import all names that a module defines with the following statement:

```
>>> from dnautil import *
>>> gc(dna)
0.5384615384615384
```

Or you can import just a few select functions from a module:

```
>>> from dnautil import gc,has_stop_codon
```

Packages

- Packages group multiple modules under one name, by using “dotted module names”. For example, the module name A.B designates a submodule named B in a package named A.

- Each package in Python is a directory which MUST contain a special file called `__init__.py`. This file can be empty, and it indicates that the directory it contains is a Python package, so it can be imported the same way a module can be imported.

Package Example

Suppose you have several modules:

- a `dnautil.py` file containing useful functions to process DNA sequences
- a `rnautil.py` file containing useful functions to process RNA sequences
- a `proteinutil.py` file containing useful functions to process protein sequences

and you want to group them in a package called `bioseq` which processes all types of biological sequences.

Package Example (cont'd)

Here's a possible structure for your package:

```
bioseq/
   __init__.py
   dnautil.py
   rnautil.py
   proteinutil.py
   ...
```

Package Example (cont'd)

You can even have other packages inside your package:

```
bioseq/
   __init__.py
   dnautil.py
   rnautil.py
   proteinutil.py
   fasta/
      __init__.py
      fastautil.py
   fastq/
      __init__.py
      fastqutil.py
   ...
```

Top-level package

Initialize the bioseq package

Subpackage for processing fasta files

Subpackage for processing fastq files

Loading From Packages

To use the module `dnautil`, we can import it in two ways:

```
>>> import bioseq.dnautil
```

Use the `dnautil` module like this:

```
>>> bioseq.dnautil.gc(dna)
```

or

```
>>> from bioseq import dnautil
```

Use the `dnautil` module like this:

```
>>> dnautil.gc(dna)
```

Loading Specific Functions From A Package

To import a specific function from a submodule in a subpackage use the following syntax:

```
>>> from bioseq.fasta.fastautil import
fastaseqread
```