SELECTED TOPICS IN ENGINEERING

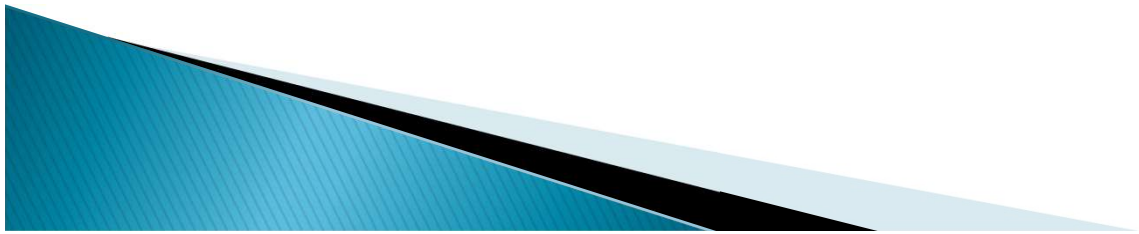
INTR. TO PROG. FOR DATA SCIENCE
ENGR 350

Tuesday-Thursday 10:00-12:45
ENG B05
2019 Summer

Dr. Banu Yobaş

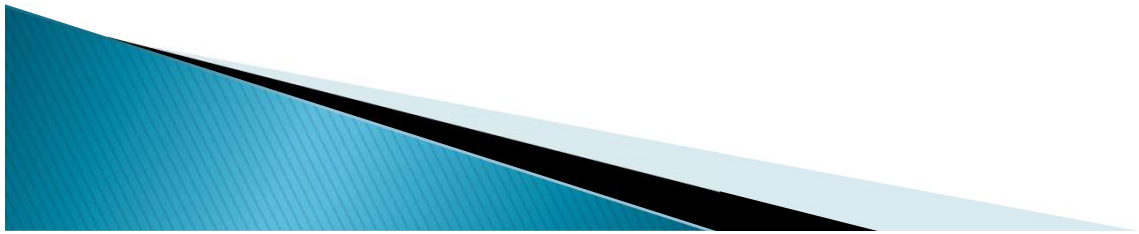
Library, Module

- A library is just a module that contains some useful definitions.



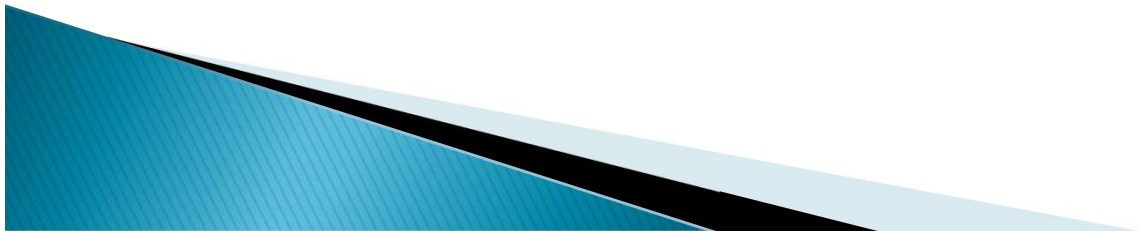
Standard Library

- Core:
  - os, sys, string, getopt, StringIO, struct, pickle, ...
- Regular expressions:
  - re module; Perl-5 style patterns and matching rules
- Internet:
  - socket, rfc822, httplib, htmllib, ftplib, smtplib, ...
- Miscellaneous:
  - pdb (debugger), profile+pstats
  - Tkinter (Tcl/Tk interface), audio, *dbm, ...



Modules

- Collection of stuff in *foo*.py file
  - functions, classes, variables
- Importing modules:
  - import re; print re.match("[a-z]+", s)
  - from re import match; print match("[a-z]+", s)
- Import with rename:
  - import re as regex
  - from re import match as m

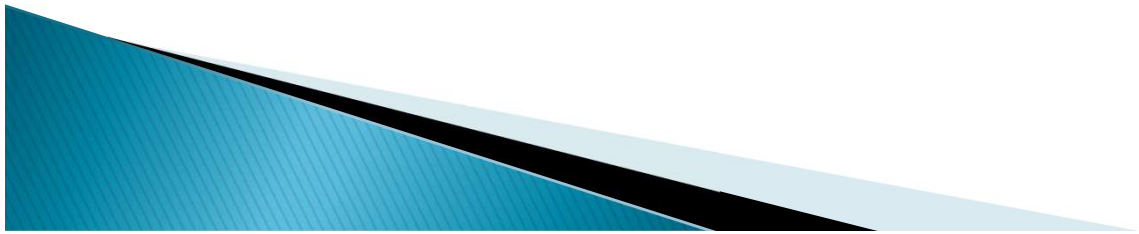


Modules

- math
- string
- random
- os
- system
- request
- urllib
- pandas
- numpy

Packages

- Collection of modules in directory
- Must have __init__.py file
- May contain subpackages
- Import syntax:
  - from P.Q.M import foo; print foo()
  - from P.Q import M; print M.foo()
  - import P.Q.M; print P.Q.M.foo()
  - import P.Q.M as M; print M.foo() # new

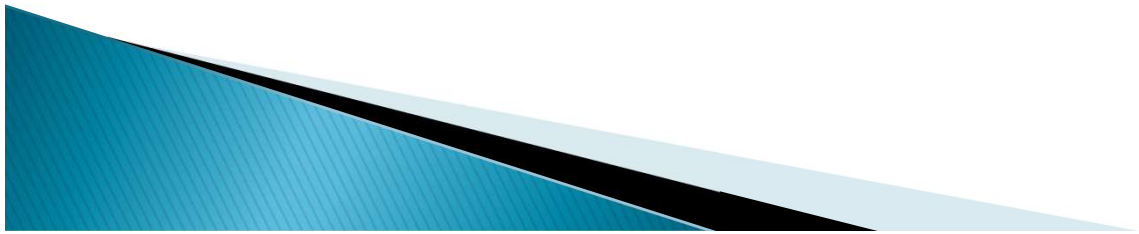


Import and reload

- import module_name
- reload module_name
- from module_name import function_name

- Naming objects differ
- All modules and objects within

IDLE -> File->Path Browser



import module vs from module import …

import sys

- This brings the name “sys” into your module.

```
sys.exit()
```

from sys import

- This does NOT bring the name “sys” into your module, instead it brings all of the names inside sys (like exit for example) into your module. Now you can access those names without a prefix, like this:

```
exit()
```

- second form will overwrite any names you have already declared — like exit say.

```
exit = 42
from sys import *       # hides my exit variable.
```

- OR more likely, the other way round

```
from sys import *
exit = 42       # now hides the sys.exit function so I can't use it.
```

import module vs from module import ... (2)

```
from os import *
from urllib import *
open(foo)
        # which open gets called, os.open or urllib.open or open?
```

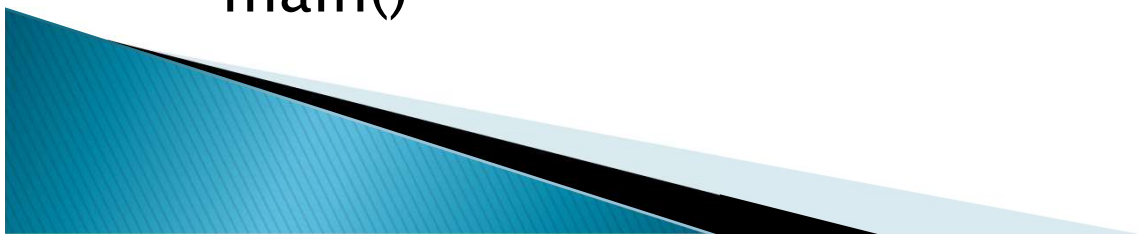
Another disadvantage of “from sys import *” is that it makes it harder to find the definition of a name.

If you see “sys.exit()” in the code, you know exactly where exit() is defined. If you see “exit()” in a module that has several “from xx import *” statements, you will have to search each imported module for the definition of exit().

__main__

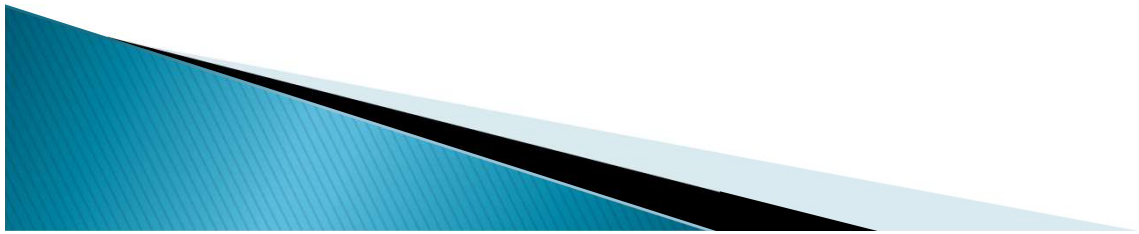
- Since Python evaluates the lines of a module during the import process, our current programs also run when they are imported into either an interactive Python session or into another Python program. Generally, it is nicer not to have modules run as they are imported. When testing a program interactively, the usual approach is to first import the module and then call its main (or some other function) each time we want to run it.
- In a program that can be *either imported (without running) or run directly, the call to main at the bottom* must be made conditional. A simple decision should do the trick.

```
if <condition>:
    main()
```



__main__

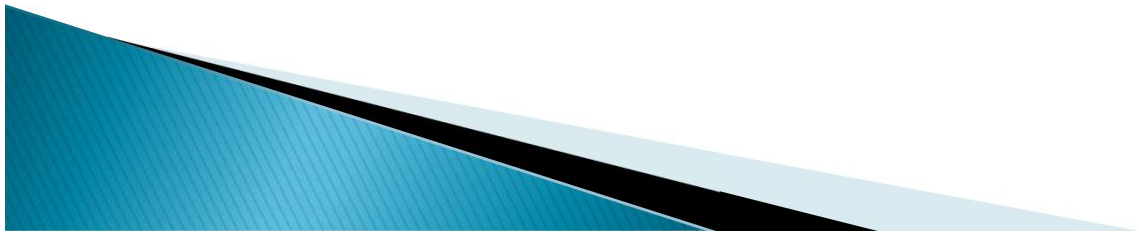
- When you create a variable outside of any function, it belongs to __main__.



__name__ of a module

- We just need to figure out a suitable condition.
- Whenever a module is imported, Python sets a special variable in the module called name to be the name of the imported module. Here is an example interaction showing what happens with the math library.

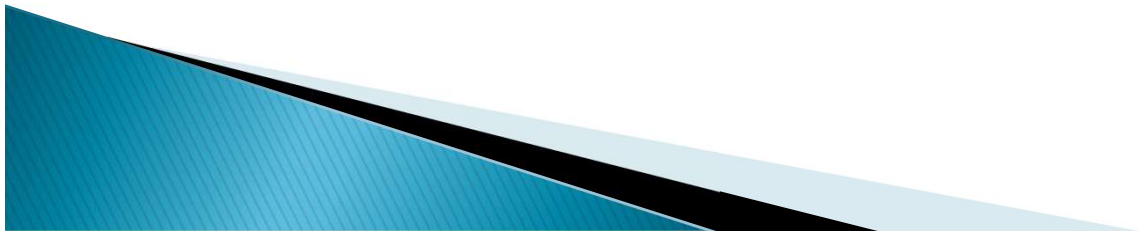
```
>>> import math
>>> math.__name__
'math'
```



__name__ of a module

- You can see that, when imported, the name variable inside the math module is assigned the string 'math'.
- However, when Python code is being run directly (not imported), Python sets the value of name to be ' main '. To see this in action, you just need to start Python and look at the value.

```
>>> __name__
'__main__'
```



__name__ of a module

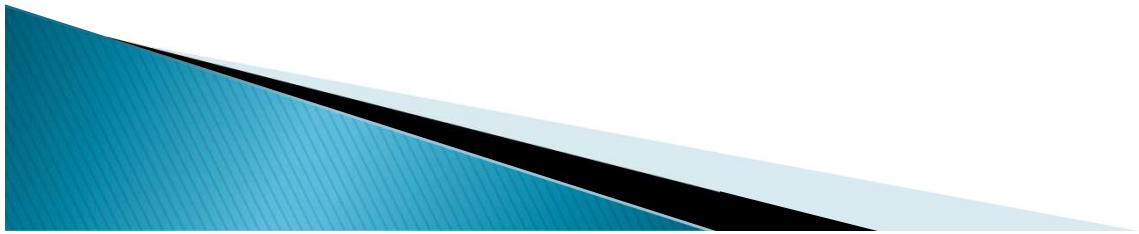
- So, if a module is imported, the code in that module will see a variable called name whose value is the name of the module. When a file is run directly, the code will see that name has the value ' main '.
- A program can determine how it is being used by inspecting this variable.

```
if __name__ == '__main__':
main()
```

- This guarantees that main will automatically run when the program is invoked directly, but it will not run if the module is imported. You will see a line of code similar to this at the bottom of virtually every Python program.

__name__ of a module

- '__main__' is the name of the scope in which top-level code executes.
- A module's __name__ is set equal to '__main__' when read from standard input, a script, or from an interactive prompt.



Accessing Web Pages – urllib

```
import urllib.request
data = urllib.request.urlretrieve(http://... )

import urllib.request
with urllib.request.urlopen('http://python.org/') as response:
    htmlpage = response.read()
type(htmlpage)

import urllib
url = 'http://www.simula.no/research/scientific/cbc'
urllib.request.urlretrieve(url, filename='webpage.html')

infile = urllib.request.urlopen(url)
type(infile)
```