Lecture 12 - Modules and the import statement

Computer Programming

Robert D. Vincent and Samia Hilal

Marianopolis College

April 10, 2022

What is a module?

- *Modules* (or *libraries*) are another method for decomposing a program.
- Any `.py` file with Python code is a *module*.
- Modules can “export” any global name (declared outside of functions with no-indentation) or function for use in other modules.
- You don’t need to take any special steps to export global names.
- A module has a name, which is just the file name (omitting the extension).

A few more builtin functions

- These functions will be useful when we learn about python standard modules.
  - `dir()` - return a “directory” of a value or object.
  - `type()` - get the type of an object.
  - `help()` - prints the documentation associated with an object.

`Example:` - The `"list"` data type in python:

```
>>> L=[]
>>> dir(list)
>>> type(L)
>>> help(list.sort) # sort method
```

Why use modules?

- Breaking your program into modules is another way to *decompose* a problem into smaller units.
- Since the code to implement a module is in a separate file, we can treat it as a black box, e.g. *abstraction*.
- Python comes with lots of interesting modules, so you don't have to write all the code yourself.
- A list of standard modules can be seen here: https://docs.python.org/3/library/

How to use modules

- Ordinarily, a Python file only has access to its own global names, and those built into Python.
- You decide which python standard module(s) your Python code will use.
- The `import` statement is used to make all or part of another module visible in your module.

import statement

Must appear before the module or name is used.

```
import modulename #imports the named modules.

# imports the selected name only
from modulename import name

# imports 'name' under the alias 'othername'
from modulename import name as othername

# imports all module members
from modulename import *
```

import examples

```
# import a single module.
import math

# import two modules
import math, numpy

# import the sine function from math
from math import sin

# import the two functions from math
from math import sin, cos

# import everything from math
from math import *
```

Creating your own module.

Create some functions to manipulate circles:

```
'''Some calculations for circles.'''
PI = 3.1414926 # 'Constant'
def area(r):
  '''Return the area of a circle with
  radius "r"'''
  return PI*r**2
def circumference(r):
  '''Return the perimeter of a circle with
  radius "r"'''
  return 2*PI*r
```

Using your own module.

```
>>> import circle
>>> dir(circle)
[’PI’, ’__builtins__’, ’__cached__’,
’__doc__’, ’__file__’, ’__loader__’,
’__name__’, ’__package__’, ’__spec__’,
’area’, ’circumference’]
>>> print(circle.PI)
3.1415926
>>> print(circle.area(1))
3.1415926
>>> print(circle.circumference(1))
6.2829852
```

What happens when a module is imported?

1. Python searches for the file in the directory containing the script, and a series of standard places.
2. Python reads the file and executes it.
3. Python creates a module object to hold the module's public functions and data.
4. Python makes the module name (or its members) available to our program.
5. If we import the module again later in the same program, we skip steps 1 through 3.

Importing Standard Modules

- Let's check the standard python module math

```
>>> dir(math)
#NameError: name ’math’ is not defined
>>> import math
>>> dir(math) #now we see
>>> help(sqrt) # yes square root is there
>>> import random # very useful module
>>> print(random.__doc__)
Random variable generators
:
>>> help(random.randint)
```

What are all those special names?

- `__doc__` Document string of the module.
- `__file__` File name of the module.
- `__name__` Module name.

```
>>> import circle
>>> print(circle.__doc__)
Some calculations for circles.
>>> print(circle.__file__)
G:\Intro-W20\L12-Modules\examples\circle.py
>>> print(circle.__name__)
circle
```

Modules are objects with values.

And those values can be assigned to other names.

```
>>> import circle
>>> x = circle    # Assign a new name.
>>> print(x.PI)
3.1415926
>>> print(x.area(1))
3.1415926
>>> print(x.circumference(1))
6.2829852
>>> print(type(circle))
<class ’module’>
```

Imported functions are objects.

We can give them local names if desired.

```
>>> import circle
>>> my_area = circle.area  # Local name.
>>> print(my_area(1))
3.1415926
>>> print(my_area(2))
12.56559704
>>> type(my_area)
<class ’function’>
```

Imported objects are not constant!

This means we can do evil things.

```
>>> import circle
>>> circle.PI = 3.0      # BAD IDEA!
>>> print(circle.area(1))
3.0
>>> print(circle.area(2))
12.0
```

Module help

```
>>> help(circle)
Help on module circle:

NAME
    circle - Some calculations for circles.

FUNCTIONS
    area(r)
        Return the area of a circle with
        radius "r"

    circumference(r)
        Return the perimeter of a circle with
        radius "r"

DATA
    PI = 3.1414926

FILE
    G:\Intro-W20\L12-Modules\examples\circle.py
```

Import individual items

- The other forms of the `import` statement let us import the names directly.
- The imported names can be used without using the module name.
- Only the imported names themselves will be available to your program.

```
>>> from circle import area, PI
>>> print(area(1) == PI)
True
>>> dir(circle)
NameError: name ’circle’ is not defined
```

Import using an assumed name

- Sometimes a name from another module might conflict with a name in your code.
- So you can import it under another name.

```
>>> from circle import area as circle_area
>>> print(circle_area(1))
3.14159
>>> print(area(1))
NameError: name ’area’ is not defined
```

Other things to note

- Only variables and functions created at the top level (no indentation) can be imported.

```
'''A trivial module.'''
def fib(n):
  '''Calculate a Fibonacci number'''
  f0,f1=0,1
  while n >= 1:
    f0,f1=f1,f0+f1
    n -= 1
  return f0
```

- Only `fib()` can be imported, `f0`, `f1`, and `n` are local to the function and not global.

Other things to note

- Modules and names can be imported within a function, making them local to the function.

```
def f(x):
    import fib
    return fib.fib(x)

x = f(10) # x = 55
print(x)
print(fib.fib(2)) # 'fib' is undefined!
```

Standard modules

Python has a **very** rich set of standard modules. These modules are part of the IDLE environment and do not need to be installed. Here is a list of common or interesting modules:

- `os` - operating system interface
- `math` - math functions
- `datetime` and `time` - get and manipulate the date (or time)
- `random` - (pseudo) random numbers
- `tkinter` - graphical user interface

os module

Contains functions and values used to access the host operating system, such as:

- `remove()` - remove a file.
- `system()` - run a command.
- `mkdir()` - create a directory.
- `chdir()` - change working directory.
- `rmdir()` - remove a directory.
- `getcwd()` - get current working directory.

os module example

```python
from os import chdir, getcwd

current_dir = getcwd()
print('Current working directory is',
      current_dir)
if current_dir.rindex('/'):
    chdir('..')
    current_dir = getcwd()
print('Working directory is now',
      current_dir)
```

`math` module

Transcendental functions:

- `sin()`, `cos()`, `tan()` - sine, cosine, tangent.
- `asin()`, `acos()`, `atan()` - inverse sine, cosine, tangent.
- `sinh()`, `cosh()`, `tanh()` - hyperbolic sine, cosine, tangent.
- `asinh()`, `acosh()`, `atanh()` - inverse hyperbolic sine, cosine, tangent.
- `ceil()`, `floor()`, `fabs()` - ceiling, floor, and absolute value.
- `exp()`, `log()`, `log10()`, `log2()` - natural exponent and logarithms.

math module example

```
from math import *
print(e, pi)       # Useful constants.
print(log(e))      # 1.0 (log base ’e’)
print(log10(100))  # 2.0 (log base 10)
print(log2(8))     # 3.0 (log base 2)
print(sin(pi / 2)) # 1.0
print(exp(1))      # ’e’
print(ceil(-1.2))  # -1
print(floor(-1.2)) # -2
print(fabs(-1.2))  # 1.2
```

Nested Standard Modules

- A major challenge associated with importing modules is nested packages.
- These packages are also called deeply nested.
- Example: datetime module

```python
import datetime.datetime.date
```

- In Python, a convenient syntax is available for importing deeply nested packages:

```python
import very.deep.module as mod.
```

- It is important to know the deep structure of objects in these modules before importing.

The `datetime` Module

Get and manipulate dates and time in Python.

- 3 main sub-modules: `datetime`, `date` and `time`
- Each sub-module has many functions.
- Use `dir()` and `help()` to know more.
- For full documentation: https://docs.python.org/3/library/datetime.html

```
>>> datetime.datetime.now() # date & time
’2020-04-26 09:26:03.478039’
>>> t= datetime.date.today() # current date.
’2020-04-26’
>>> d= datetime.date(2020,4,22) # date object
>>> t > d   # returns True
```

The `time` Module

Use different time-related functions.

- `time.time()` - returns the time in seconds since the epoch (platform-dependent variable, indicates the beginning time of a system).
- `time.ctime()` - convert a time expressed in seconds since the epoch to a string.
- `time.sleep(s)` - delay execution for s seconds

```
>>> time.ctime(time.time()) #current time
'Sat Apr 25 08:07:37 2020'
>>> time.sleep(2) # pause for 2 seconds
```

datetime module example

```python
from datetime import date

# Prints days left for my Birthday
today = date.today()
print("Today is", today)
my_bday = date(today.year, 5, 25)
if my_bday < today: #my birthday has passed
    my_bday = my_bday.replace(
        year=today.year + 1) #next year then
time_diff = my_bday - today
print(my_bday, "is", time_diff.days,
      "days from today")
```

`random` module

Pseudorandom number generation. Useful for `"luck of the draw"` games and games where the computer generates (or chooses) a random value.

Useful functions include:

- `randint(a, b)` - return random integer $x, a \leq x \leq b$
- `shuffle(L)` - rearrange list L in random order.
- `random()` - return random float $x, 0 \leq x < 1$
- `choice(L)` - return a random item from list L.

random module examples

```
>>> from random import *
>>> random()
0.6876409240484214
>>> random()
0.5700914449200836
>>> randint(1, 100) # prints 60
>>> randint(1, 100) # prints 85
>>> x = list(range(10))
>>> print(x)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> shuffle(x)
>>> print(x)
[9, 3, 7, 8, 0, 1, 2, 6, 4, 5]
>>> choice(x) # prints 4
>>> choice(x) # prints 0
```

`tkinter` module example

Very complicated, but allows you to create graphical interfaces.

```python
import tkinter
wnd = tkinter.Tk()
wnd.title('A simple example')

lbl = tkinter.Label(wnd,
        text='This is a very simple GUI')
lbl.pack()
btn = tkinter.Button(wnd,
        text="Cancel", command=wnd.quit)
btn.pack()
wnd.mainloop()
```

Import Statement Summary

Summary of import options & object reference:
Using math module as example.

```python
# import the module
import math   # use math.sin or math.cos

#import all objects - not recommended
from math import *   # use sin or cos

from math import sin, cos #use sin & cos only

# Avoid name conflicts-alias imported objects
# sin & cos not defined, use m_sin & m_cos
from math import sin as m_sin,cos as m_cos
```

Summary

- We can decompose programs into functions and also into *modules*.
- In Python, a module is most typically a `.py` file.
- Your program can use global names defined in other files by using the `import` statement.
- There are many standard modules in Python. You are not required to memorize them.
- Use them as reference and be sure you understand how to use `import`!