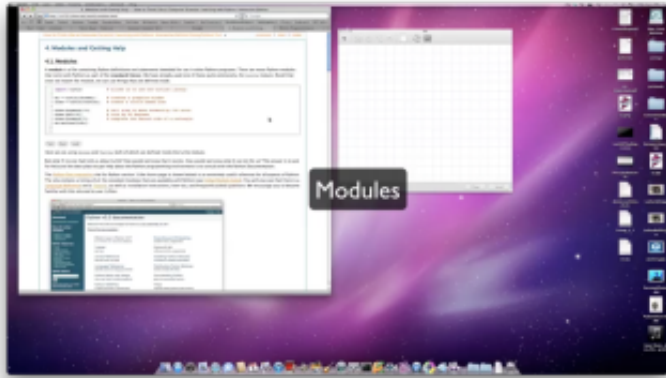


Modules and Getting Help

Modules

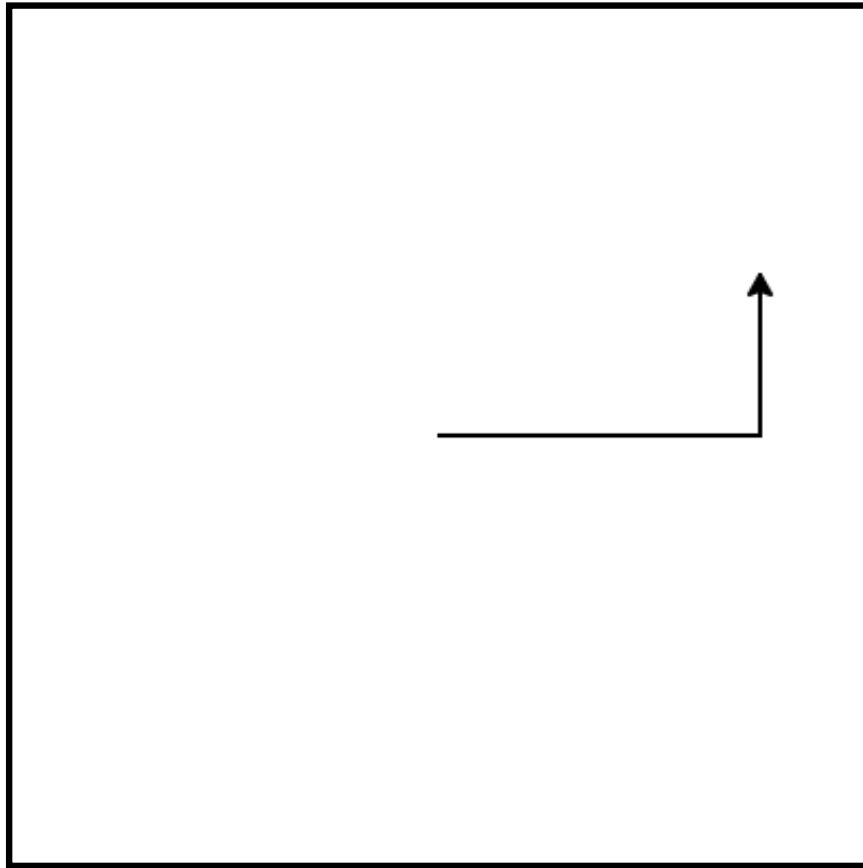


A **module** is a file containing Python definitions and statements intended for use in other Python programs. There are many Python modules that come with Python as part of the **standard library**. We have already used one of these quite extensively, the `turtle` module. Recall that once we import the module, we can use things that are defined inside.

```
1 import turtle                # allows us to use the turtles library
2
3 wn = turtle.Screen()         # creates a graphics window
4 alex = turtle.Turtle()       # create a turtle named alex
5
6 alex.forward(150)            # tell alex to move forward by 150 units
7 alex.left(90)                # turn by 90 degrees
8 alex.forward(75)             # complete the second side of a rectangle
9 wn.exitonclick()
10
```

ActiveCode: 1 (chmod_01)

Run



Here we are using `Screen` and `Turtle` , both of which are defined inside the `turtle` module.

But what if no one had told us about `turtle`? How would we know that it exists. How would we know what it can do for us? The answer is to ask for help and the best place to get help about the Python programming environment is to consult with the Python Documentation.

The Python Documentation (<http://docs.python.org/py3k/>) site for Python version 3 (the home page is shown below) is an extremely useful reference for all aspects of Python. The site contains a listing of all the standard modules that are available with Python (see Global Module Index (<http://docs.python.org/py3k/py-modindex.html>)). You will also see that there is a Language Reference (<http://docs.python.org/py3k/reference/index.html>) and a Tutorial (<http://docs.python.org/py3k/tutorial/index.html>), as well as installation instructions, how-tos, and frequently asked questions. We encourage you to become familiar with this site and to use it often.



If you have not done so already, take a look at the Global Module Index. Here you will see an alphabetical listing of all the modules that are available as part of the standard library. Find the turtle module.

Python Module Index — Python v3.2 documentation

http://docs.python.org/py3k/py-modindex.html

Apple Yahoo! Amazon Google Google Maps YouTube Wikipedia News (377) Popular My Favorites RentalLocalExpert Nottingham China Android RST Info

Python v3.2 documentation » modules | index

Quick search

Go

Enter search terms or a module, class or function name.

Python Module Index

[_](#) [a](#) [b](#) [c](#) [d](#) [e](#) [f](#) [g](#) [h](#) [i](#) [j](#) [k](#) [l](#) [m](#) [n](#) [o](#) [p](#) [q](#) [r](#) [s](#) [t](#) [u](#) [v](#) [w](#) [x](#) [y](#) [z](#)

—

`__future__` Future statement definitions

`__main__` The environment where the top-level script is run.

`_dummy_thread` Drop-in replacement for the `_thread` module.

`_thread` Low-level threading API.

a

`abc` Abstract base classes according to PEP 3119.

`aifc` Read and write audio files in AIFF or AIFC format.

`argparse` Command-line option and argument-parsing library.

`array` Space efficient arrays of uniformly typed numeric values.

`ast` Abstract Syntax Tree classes and manipulation.

`asynchat` Support for asynchronous command/response protocols.

`asynore` A base class for developing asynchronous socket handling services.

`atexit` Register and execute cleanup functions.

`audioop` Manipulate raw audio data.

b

`base64` RFC 3548: Base16, Base32, Base64 Data Encodings

`bdb` Debugger framework.

`binascii` Tools for converting between binary and various ASCII-encoded binary representations.

`binhex` Encode and decode files in binhex4 format.

`bisect` Array bisection algorithms for binary searching.

`builtins` The module that provides the built-in namespace.

`bz2` Interface to compression and decompression routines compatible with bzip2.

c

`calendar` Functions for working with calendars, including some emulation of the Unix `cal` program.

`cgi` Helpers for running Python scripts via the Common Gateway Interface.

`cgihttp` Configurable traceback handler for CGI scripts.

`chunk` Module to read IFF chunks.

`cmath` Mathematical functions for complex numbers.

`cmd` Build line-oriented command interpreters.

`code` Facilities to implement read-eval-print loops.

`codecs` Encode and decode data and streams.

`codeop` Compile (possibly incomplete) Python code.

`collections` Container datatypes

`coloursys` Conversion functions between RGB and other color systems.

`compileall` Tools for byte-compiling all Python source files in a directory tree.

concurrent

`configparser` Configuration file parser.

`contextlib` Utilities for with-statement contexts.

`copy` Shallow and deep copy operations.

`copyreg` Register pickle support functions.

`cProfile` Python profiler

`crypt (Unix)` The `crypt()` function used to check Unix passwords.

`csv` Write and read tabular data to and from delimited files.

`ctypes` A foreign function library for Python.

23.1. turtle — Turtle graphics — Python v3.2 documentation

http://docs.python.org/py3k/library/turtle.html#module-turtle

Python v3.2 documentation » The Python Standard Library » 23. Program Frameworks »

23.1. turtle — Turtle graphics

23.1.1. Introduction

Turtle graphics is a popular way for introducing programming to kids. It was part of the original Logo programming language developed by Wally Feurzig and Seymour Papert in 1966.

Imagine a robotic turtle starting at (0, 0) in the x-y plane. Give it the command `turtle.forward(15)`, and it moves (on-screen!) 15 pixels in the direction it is facing, drawing a line as it moves. Give it the command `turtle.left(25)`, and it rotates in-place 25 degrees clockwise.

By combining together these and similar commands, intricate shapes and pictures can easily be drawn.

The `turtle` module is an extended reimplementation of the same-named module from the Python standard distribution up to version Python 2.5.

It tries to keep the merits of the old turtle module and to be (nearly) 100% compatible with it. This means in the first place to enable the learning programmer to use all the commands, classes and methods interactively when using the module from within IDLE run with the `-i` switch.

The turtle module provides turtle graphics primitives, in both object-oriented and procedure-oriented ways. Because it uses `tkinter` for the underlying graphics, it needs a version of Python installed with Tk support.

The object-oriented interface uses essentially two+two classes:

1. The `TurtleScreen` class defines graphics windows as a playground for the drawing turtles. Its constructor needs a `tkinter.Canvas` or a `ScrolledCanvas` as argument. It should be used when `turtle` is used as part of some application.

The function `screen()` returns a singleton object of a `TurtleScreen` subclass. This function should be used when `turtle` is used as a standalone tool for doing graphics. As a singleton object, inheriting from its class is not possible.

All methods of `TurtleScreen/Screen` also exist as functions, i.e. as part of the procedure-oriented interface.

2. `RawTurtle` (alias: `RawPen`) defines Turtle objects which draw on a `TurtleScreen`. Its constructor needs a `Canvas`, `ScrolledCanvas` or `TurtleScreen` as argument, so the `RawTurtle` objects know where to draw.

Derived from `RawTurtle` is the subclass `Turtle` (alias: `Pen`), which draws on "the" `Screen` instance which is automatically created, if not already present.

All methods of `RawTurtle/Turtle` also exist as functions, i.e. part of the procedure-oriented interface.

The procedural interface provides functions which are derived from the methods of the classes `Screen` and `Turtle`. They have the same names as the corresponding methods. A `Screen` object is automatically created whenever a function derived from a `Screen` method is called. An (unnamed) turtle object is automatically created whenever any of the functions derived from a `Turtle` method is called.

To use multiple turtles on a screen one has to use the object-oriented interface.

Note: In the following documentation the argument list for functions is given. Methods, of course, have the additional first argument `self` which is omitted here.

Turtle star

Turtle can draw intricate shapes using programs that repeat simple moves.



```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
    forward(200)
    left(170)
    if abs(pos()) < 1:
        break
end_fill()
done()
```

You can see that all the turtle functionality that we have talked about is there. However, there is so much more. Take some time to read through and familiarize yourself with some of the other things that turtles can do.

Note: Python modules and limitations with activecode

Throughout these lectures, activecode windows allow you to practice the Python that you are learning. We mentioned in the first chapter that programming is normally done using some type of development environment and that the activecode used here was strictly to help us learn. It is not the way we write production programs.

To that end, it is necessary to mention that many of the modules available in standard Python will **not** work in the activecode environment. In fact, only turtle, math, and random have been ported at this point. If you wish to explore any additional modules, you will need to also explore using a more robust development environment.

Check your understanding

mod-1: In Python a module is:

- ☒ a) A file containing Python definitions and statements intended for use in other Python programs.
- ☐ b) A separate block of code within a program.
- ☐ c) One line of code in a program.
- ☐ d) A file that contains documentation about functions in Python.

Check Me

Compare Me

Correct!! A module can be reused in different programs.

mod-2: To find out information on the standard modules available with Python you should:

- ☒ a) Go to the Python Documentation site.
- ☐ b) Look at the import statements of the program you are working with or writing.
- ☐ c) Ask the professor
- ☐ d) Look in this lecture.

Check Me

Compare Me

Correct!! The site contains a listing of all the standard modules that are available with Python.

mod-3: True / False: All standard Python modules will work in activecode.

- ☐ a) True
- ☒ b) False

Check Me

Compare Me

Correct!! Only turtle, math, and random have been ported to work in activecode at this time.

© Copyright 2013 Brad Miller, David Ranum, Created using Runestone Interactive.