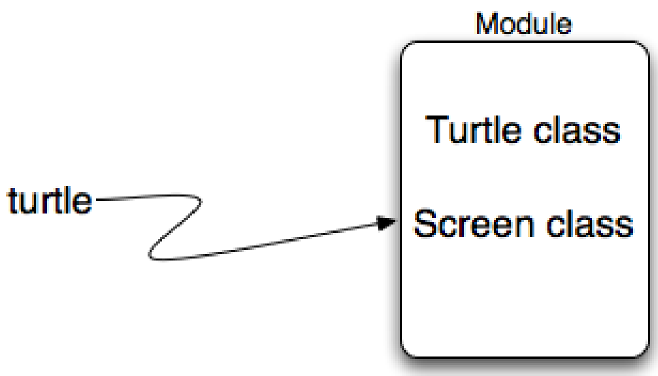
More About Using Modules

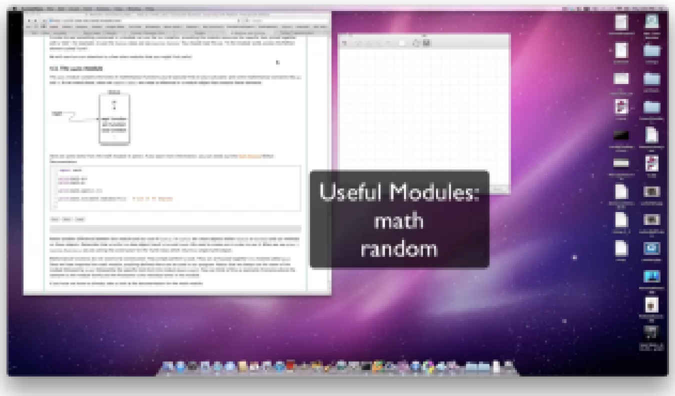
Before we move on to exploring other modules, we should say a bit more about what modules are and how we typically use them. One of the most important things to realize about modules is the fact that they are data objects, just like any other data in Python. Module objects simply contain other Python elements.

The first thing we need to do when we wish to use a module is perform an `import`. In the example above, the statement `import turtle` creates a new name, `turtle`, and makes it refer to a module object. This looks very much like the reference diagrams we saw earlier for simple variables.



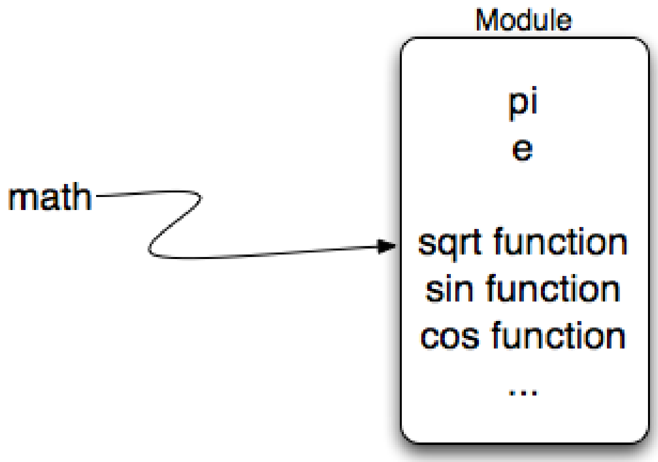
In order to use something contained in a module, we use the dot notation, providing the module name and the specific item joined together with a “dot”. For example, to use the `Turtle` class, we say `turtle.Turtle`. You should read this as: “In the module turtle, access the Python element called Turtle”.

We will now turn our attention to a few other modules that you might find useful.



The math module

The `math` module contains the kinds of mathematical functions you would typically find on your calculator and some mathematical constants like pi and e. As we noted above, when we `import math`, we create a reference to a module object that contains these elements.



Here are some items from the math module in action. If you want more information, you can check out the Math Module (http://docs.python.org/py3k/library/math.html#module-math) Python Documentation.

```
import math

print(math.pi)
print(math.e)

print(math.sqrt(2.0))

print(math.sin(math.radians(90)))    # sin of 90 degrees

```

ActiveCode: 1 (chmodule_02)

Run

```
3.14159265359
2.71828182846
1.41421356237
1.0
```

Notice another difference between this module and our use of `turtle` . In `turtle` we create objects (either `Turtle` or `Screen`) and call methods on those objects. Remember that a turtle is a data object (recall `alex` and `tess`). We need to create one in order to use it. When we say `alex = turtle.Turtle()` , we are calling the constructor for the Turtle class which returns a single turtle object.

Mathematical functions do not need to be constructed. They simply perform a task. They are all housed together in a module called math. Once we have imported the math module, anything defined there can be used in our program. Notice that we always use the name of the module followed by a dot followed by the specific item from the module (`math.sqrt`). You can think of this as lastname.firstname where the lastname is the module family and the firstname is the individual entry in the module.

If you have not done so already, take a look at the documentation for the math module.

Check your understanding

mod-4: Which statement allows you to use the math module in your program?

[✓ a) import math]

b) include math

c) use math

d) You don’t need a statement. You can always use the math module

Check Me Compare Me

Correct!! The module must be imported before you can use anything declared inside the module.

The random module

We often want to use **random numbers** in programs. Here are a few typical uses:

- To play a game of chance where the computer needs to throw some dice, pick a number, or flip a coin,
- To shuffle a deck of playing cards randomly,
- To randomly allow a new enemy spaceship to appear and shoot at you,
- To simulate possible rainfall when we make a computerized model for estimating the environmental impact of building a dam,
- For encrypting your banking session on the Internet.

Python provides a module `random` that helps with tasks like this. You can take a look at it in the documentation. Here are the key things we can do with it.

```
import random

prob = random.random()
print(prob)

diceThrow = random.randrange(1,7)       # return an int, one of 1,2,3,4,5,6
print(diceThrow)

```

ActiveCode: 2 (chmodule_rand)

Run

```
0.787489397691
2
```

Press the run button a number of times. Note that the values change each time. These are random numbers.

The `randrange` function generates an integer between its lower and upper argument, using the same semantics as `range` — so the lower bound is included, but the upper bound is excluded. All the values have an equal probability of occurring (i.e. the results are *uniformly* distributed).

The `random()` function returns a floating point number in the range [0.0, 1.0) — the square bracket means "closed interval on the left" and the round parenthesis means "open interval on the right". In other words, 0.0 is possible, but all returned numbers will be strictly less than 1.0. It is usual to *scale* the results after calling this method, to get them into a range suitable for your application.

In the case shown here, we've converted the result of the method call to a number in the range [0.0, 5.0). Once more, these are uniformly distributed numbers — numbers close to 0 are just as likely to occur as numbers close to 0.5, or numbers close to 1.0. If you continue to press the run button you will see random values between 0.0 and up to but not including 5.0.

```
import random

prob = random.random()
result = prob * 5
print(result)

```

ActiveCode: 3 (chmodule_rand2)

Run

```
4.61885398305
```

It is important to note that random number generators are based on a **deterministic** algorithm — repeatable and predictable. So they're called **pseudo-random** generators — they are not genuinely random. They start with a *seed* value. Each time you ask for another random number, you'll get one based on the current seed attribute, and the state of the seed (which is one of the attributes of the generator) will be updated. The good news is that each time you run your program, the seed value is likely to be different meaning that even though the random numbers are being created algorithmically, you will likely get random behavior each time you execute.

Lab

- Sine Wave (http://dcs.asu.edu/faculty/abansal/CST100/Labs/PythonModules-PlottingSineWaveLab.html) In this guided lab exercise we will have the turtle plot a sine wave.

Check your understanding

mod-5: Which of the following is the correct way to reference the value pi within the math module. Assume you have already imported the math module.

[✓ a) math.pi]

b) math(pi)

c) pi.math

d) math->pi

Check Me   Compare Me

Correct!! To invoke or reference something contained in a module you use the dot (.) notation.

mod-6: Which module would you most likely use if you were writing a function to simulate rolling dice?

a) the math module

[✓ b) the random module]

c) the turtle module

d) the game module

Check Me | Compare Me

Correct!! You would likely call the function random.randrange.

mod-7: The correct code to generate a random number between 1 and 100 (inclusive) is:

[✓ a) prob = random.randrange(1, 101)]

b) prob = random.randrange(1, 100)

c) prob = random.randrange(0, 101)

d) prob = random.randrange(0, 100)

Check Me | Compare Me

Correct!! This will generate a number between 1 and 101, but does not include 101.

mod-8: One reason that lotteries don't use computers to generate random numbers is:

a) There is no computer on the stage for the drawing.

[✓ b) Because computers don't really generate random numbers, they generate pseudo-random numbers.]

c) They would just generate the same numbers over and over again.

d) The computer can't tell what values were already selected, so it might generate all 5's instead of 5 unique numbers.

Check Me | Compare Me

Correct!! Computers generate random numbers using a deterministic algorithm. This means that if anyone ever found out the algorithm they could accurately predict the next value to be generated and would always win the lottery.

Note

This workspace is provided for your convenience. You can use this activecode window to try out anything you like.

ActiveCode: 4 (scratch_04)

Run