# 11:Modules

## Modules in Python

- Python subprograms can be collected into independent modules
- These modules can be imported in other programs
- This means, that you can reuse the code you have written in other programs without needing to copy-paste it.

## Useful modules in Python

- Python comes with a set of useful modules you can use in your own programs.

- We've already used some, including e.g. `math`, `string` and `__future__`.

## Useful modules (cont.)

- A complete list about global modules in Python 2.7 can be found here:

http://docs.python.org/2.7/py-modindex.html

## Using modules

- To use a module, it needs to be imported
- The syntax for importing the whole module:

```
import module_name
```

- ...for example:

```
import math
```

## Using modules (cont.)

- When a module is imported, you can call any function or procedure from module by specifying the name of the module and a subroutine:

```
import math
print math.sqrt(49) # square root for 49
```

## Importing routines from module

- If you only need a couple of routines from the module, you can specify them in the import statement. Use syntax

```
from module import routinelist
```

- Routinelist specifies any number of routines, separated with commas

## Importing routines (cont.)

- For example, to import routines `sqrt` and `log10` from module math, use

```
from math import sqrt, log10
```

- Or, to import routine `capwords` from module string, use

```
from string import capwords
```

## Importing routines (cont.)

- When a specified routine is imported from a module, you don't need to use module name when calling the routine:

```
from math import sqrt
print sqrt(49)
```

- …or…

```
import math
print math.sqrt(49)
```

## Naming imported routines

- If you have imported the whole module (using import module statement), but are going to use a certain routine a lot, you can assign it a local name:

```
myFunctionName = module.routine
```

## Naming imported routines (cont.)

- When the routine is named, you can use that name to call the routine:

```
import math

square = math.sqrt
print square(100) # outputs 10.0

lg = math.log10
print lg(100) # outputs 2.0
```

## Importing all routines from a module

- An asterisk (*) can be used to import all routines from a module:

```
from module_name import *
```

- This means, that you can call any routine from that module without using the module name. However, this has a tendency to lead to unreadable code, so it usually should be avoided.

## Importing all routines... (cont.)

- To import all routines from math, use

```
from math import *
```

- Now you can call any routine from math module without specifying the module name:

```
print sqrt(250)
print sin(90)
print pi
```

## Defining own modules

- Any own program with subroutines (procedures or functions) can be used as a module.

- Program file's name is used as a module name.

- Note, that again Python interpreter looks at the *current folder* for modules.

## Defining own modules (cont.)

- For example, file my_module.py:

```
def add(a, b, c):
   sum = a + b + c
   return sum

def multi(a, b, c):
   return a * b * c
```

# Defining own modules (cont.)

- Using my_module.py:

```
import my_module #note, that no need for .py
print my_module.add(1,2,3) # output 6

from my_module import multi
print multi (2,2,2) # output 8

from my_module import *
print multi(1,2,3) + add(2,3,4) # output 15
```

## Reloading a module

- If you make changes to your own module, and need to use updated version when old version is already loaded, you can reload the module by using the reload statement:

```
reload(my_module)
```

## Bit more about function parameters

- Especially, when using routines in modules (but in other use too), it is often handy to give some subprogram parameters default values.

- Parameters with default values are optional when calling the subprogram. If the value is not defined in the subprogram call, the default value will be used.

## Default parameter values

- The default value for parameter can be assigned just like any other variable:

```
def myProcedure(a=3):
   print a

myProcedure(5) # output 5
myProcedure() # output 3
```

## Default parameter values (cont.)

- It is possible to define more than one default value:

```
def createPerson(name="John", age=21):
   return (name, age)

print createPerson() # output ('John', 21)
print createPerson("James") #output ('James', 21)
print createPerson("Jane",24) #output ('Jane', 24)
print createPerson(age=33) #output ('John',33)
```

## Default parameter values (cont.)

- It is also possible to mix default values with non-default values; the parameters with no default value are mandatory when calling the subprogram:

```
def multiplyString(str, number=2):
    return str * number

print multiplyString("aa") # output 'aaaa'
print multiplyString("bb",3) # output 'bbbbbb'
```

## Example

- A function that reads the contents of a file into a list, and removes line feeds from each line if wanted:

```
def fileToList(filename, removeLF = False):
   f = open (filename, "r")
   lst = f.readlines()
   if removeLF:
       for index, line in enumerate(lst):
         lst[index] = line.replace("\n","")
   f.close()
   return lst

m1 = fileToList("file.txt") # doesn't remove line feeds
m2 = fileToList("file.txt", True) # removes lf:s
```

## Using module as standalone

- It's quite typical for the program acting as a module to include a main program as well

- If such program is imported, the main program is executed at import

- This may lead to potential problems

## Example

- Consider a program test_module.py:

```
def test():
    print "Hello!"

# main program starts after functions
print "Hi!"
```

## Example (cont.)

- Test drive:

```
>>> import test_module
Hi!
>>> test_module.test()
Hello!
```

# The __main__ variable

- It is possible to check if the code is executed as a module or as a standalone program.
- The trick is to check if the __main__ variable's value is "__main__"
- If True, the program is executed standalone, if False, as a module

## Example

- Fixed example program:

```
def test():
    print "Hello!"

if __name__ == "__main__":
    # Main program goes here
    print "Hi!"
```

## Example (cont.)

- Example run:

```
>>> import test_module
>>> test_module.test()
Hello!
```

## Back to modules

- Finally, let's have a look at some of Pythons global modules that can be used in own programs.

## Module random

- Module for generating random numbers. Some operations:

  `random.randint(a,b)` – returns a random number between **a** and **b**, including end points

  `random.choice(seq)` – returns a random item from sequence seq

## Example

- Function that simulates a number of dice throws:

```
def throwDices(number):
    lst = []
    for i in xrange(number):
        lst.append(random.randint(1,6))
    return lst
```

# Module fractions

- Module for implementing fraction numbers in Python. Some operations:

```
from fractions import Fraction
print Fraction(3,6) # output 1/2 (3/6 == ½)
res = Fraction(1,2) + Fraction(1,4)
print res # output 3/4
myFrac = Fraction(2,8)
print myFrac # output 1/4
print myFrac.numerator # outputs 1
print myFrac.denominator # outputs 4
print Fraction(0.75) # output 3/4
```

## Module math

- Module for various mathematical operations. Some operations:

  `math.fsum(seq)` – returns a floating point sum of all items in sequence seq
  `math.pi` – returns π
  `math.sin(x), math.cos(x), math.tan(x)` – return sine, cosine or tangent of x radians

## Module string

- Module for various string operations. Some constants:

  `string.ascii_letters` – returns a string with all letters a...z + A...Z
  `string.ascii_lowercase` and `string.ascii_uppercase` – return only lowercase or uppercase letters
  `string.digits` – returns a string with all digits 0...9
  See also formatting strings, which is useful for many occasions.

## Module datetime

- Module for various date and time functions. Example of some operations:

```
from datetime import date
d = date.today()
print d.day # output current day number
print d.month # output current month number
print d.year # output current year
otherDate = date(2005, 6, 23) # 23.6.2005
diff = d – otherDate
print diff.days # days between dates
```

## Module datetime (cont.)

- Class datetime is useful if you want to utilize time with the date:

```
from datetime import datetime
# In order: year, month, day, hour, minutes, seconds
d = datetime(2016, 22, 11, 15, 15, 0)
d2 = datetime(2016, 22, 11, 16, 15, 0)
diff = d2 – d1
print diff.seconds # 3600 == one hour
```

## Class timedelta

- The timedelta class is useful when you want to roll the date forwards or backwards.

```
from date import date, timedelta
d1 = date(2016, 11,22)
diff = timedelta(10) # 10 days
d2 = d1 + diff
print d2 # Outputs 2016-12-02
```

## Class timedelta (cont.)

- Timedelta accepts different time formats:

```
dif1 = timedelta(4) # 4 days
dif2 = timedelta(hours = 5) # 5 hours
dif3 = timedelta(seconds = 44) # 44 seconds

dif4 = timedelta(days = 3, minutes = 10, seconds = 3)
print dif4.seconds # 603! Not including year/month/day
print dif4.days # 1
```

## Class timedelta (cont.)

- Note, that though you can use weeks, hours, minutes and seconds to initalize a timedelta object, you can only return **days** or **seconds**.

- These are hence NOT supported:

```
td = timedelta(days = 2, hours = 3)
td.years # DOES NOT WORK
td.months # DOES NOT WORK
td.hours # DOES NOT WORK
td.minutes # DOES NOT WORK
```

## Example 1

- Query user for a string, then output 5 random letters from that string:

```
import random
str = raw_input("Give a string:")
for i in range(5):
   print random.choice(str)
```

## Example 2

- Query user for birthday, then display the number of days until next birthday

```
from datetime import date
mon = input("Month of birth (1-12):")
day = input("Day of birth (1-31):")
today = date.today()
# see if no birthday yet this year
if date(today.year, mon, day) > today:
   diff = date(today.year, mon, day) – today
else:
   diff = date(today.year + 1, mon, day) – today
print "Days until next birthday", diff.days
```

## Example 3

- Calculate the average of numbers in list by using the math module:

```
import math

def average(listOfNumbers):
    sum = math.fsum(listOfNumbers)
    return sum / len(listOfNumbers)
```