Hello Spam!

PYTHON BASICS

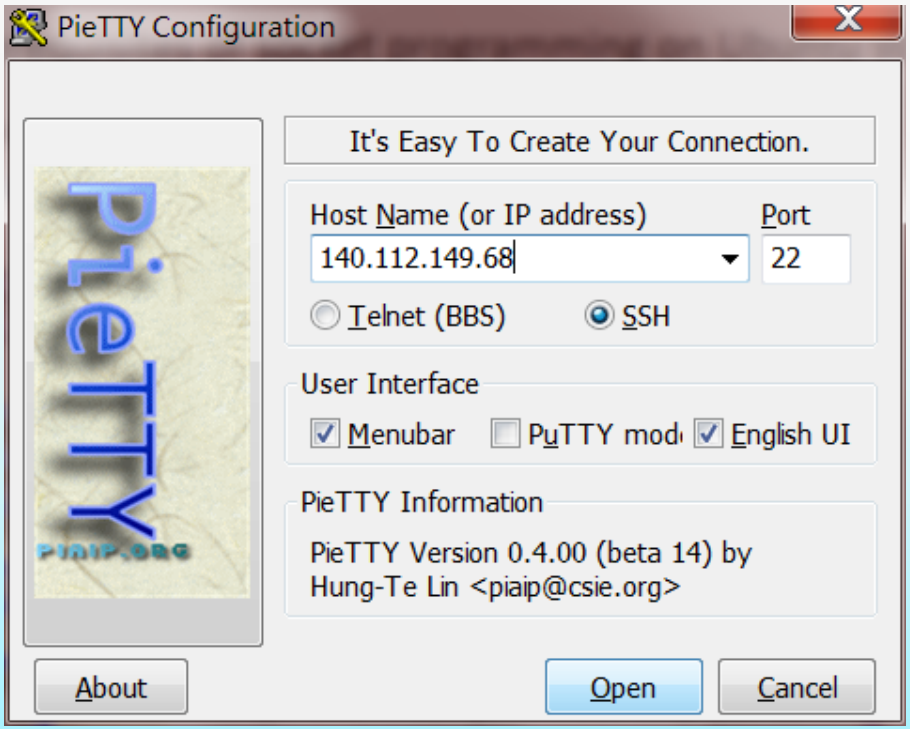
Jui Pann

Eggs

- Prerequisites
- Hello Spam!
- Data Types
- Variables, Operators, and Value Assignment
- Dynamic Typing
- String, List, Dictionary, Tuple
- Statements
- Odds-and-Ends
- Python Scoping and Function
- Python Module
- OOP, Class, Object

Prerequisites

- Download pietty for establishing ssh connections.
- http://ntu.csie.org/~piaip/pietty/



UNIX commands

- Basic instructions
  - **passwd**: change password
  - **exit**: logout
  - **ps**: show processes
  - ps aux | grep xxx: show processes with keyword

UNIX commands

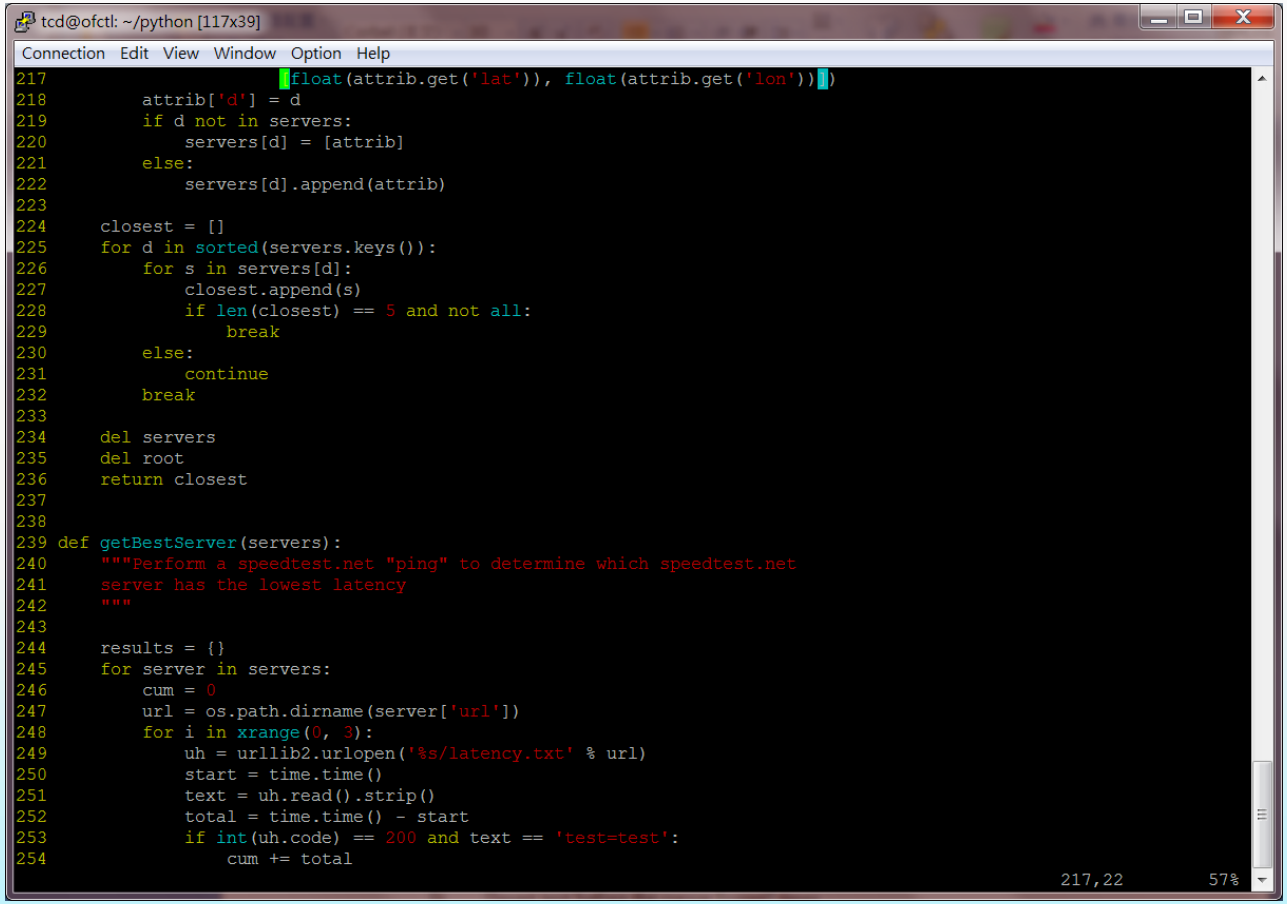
- Useful commands
  - **man**: manual page
  - **ls**: list the contents of current folder
  - **ls -a**: list all (including the hidden files)
  - **ls –aux** : list all files with detailed information
  - **cp** : copy files from one folder/directory to another
  - **cp –r** : copy the whole folder to another
  - **mkdir** : create a folder
  - **rmdir**: delete a folder
  - **pwd** : display your current path

UNIX commands

- More useful commands
  - **cd** : change folder
  - **rm** : delete files
  - **rm –r** : remove the whole folder
  - **tar** : pack and/or compress files
  - **mv** : move or rename file
  - **exit** : logout
  - **cat**: read a file
  - **more**: read a file with scrolling by enter or space key
  - **less**: read a file with scrolling by PgUp or PgDn

UNIX Editors

- nano
- vi
- vim
- gedit
- joe
- emacs



vim commands

- To open or edit a file:

```
tcd@ofctl:~/python$ vim test.py
```

- Insert text: i
- Move the cursor: h (←), j (↓), k (↑), l (→), PgUp, PgDn, Home (0), End($).
- Move n letters: n[space]
- Move n lines: n[enter]

vim commands

- Move to the head of the file: gg
- Move to the tail of the file: G
- Move to the nth line of the file: nG
- Delete a letter: x
- Delete n letters: nx
- Delete a line: dd
- Delete n lines: ndd

vim commands

- Delete lines to the tail: dG
- Delete lines to the head: dgg
- Delete letters to the head of line: do
- Delete letters to the tail of line: d$
- Copy the line: yy
- Copy n lines: nyy
- Copy lines to the tail: yG
- Copy lines to the head: ygg

vim commands

- Copy letters to the head of line: yo
- Copy letters to the tail of line: y$
- Paste on the next/last line: p/P
- Merge two lines: J
- Undo: u
- Redo: [ctrl]+r
- Repeat the last action: .

vim commands

- i: enter INSERT mode at the cursor position
- a: enter INSERT mode at the next position of the cursor
- o: enter INSERT mode at the next newline of cursor position
- O: enter INSERT mode at the last newline of cursor position
- Esc: enter VIEW mode

vim commands

- :w write the file
- :q simply quit vim
- :q! quit vim without saving
- :wq save and quit vim
- :n1,n2s/word1/word2/g
  - Replace word1 with word2 from line n1 to line n2
- :1,$s/word1/word2/g
  - Replace word1 with word2 from head to tail of file

Eggs

- Prerequisites
- Hello Spam!
- Data Types
- Variables, Operators, and Value Assignment
- Dynamic Typing
- String, List, Dictionary, Tuple
- Statements
- Odds-and-Ends
- Python Scoping and Function
- Python Module
- OOP, Class, Object

Python

- Proposed by Guido van Rossum at 1989.
- Python originally means:

Python Philosophy

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts!

Why Python?

- Not hard to read, write and maintain
- No compiling or linking: short development cycle
- No type declarations: simpler, shorter, flexible
- Automatic memory management: 3R
- OOP: code structuring and reusability
- Interactive shell interface: in-time execution results
- Classes, modules: programming-in-the-large support
- Famous: Google, YouTube, NASA, NSA, Pixar, etc

Hello Spam!

- Enter Python Shell

```
tcd@ofctl:~/python$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Hello Spam!

```
>>> print 'Hello Spam!'
Hello Spam!
```

.py script(file)

- Create a .py file

```
tcd@ofctl:~/python$ vim hello.py
```

- Edit the file

```
1
2 x = 'Hi Eggs, My name is Spam.'
3 print 'Spam:',x
4 print 'Eggs:','Hello Spam!'
5
6 print 1+2
7 print 2 ** 20
8
```

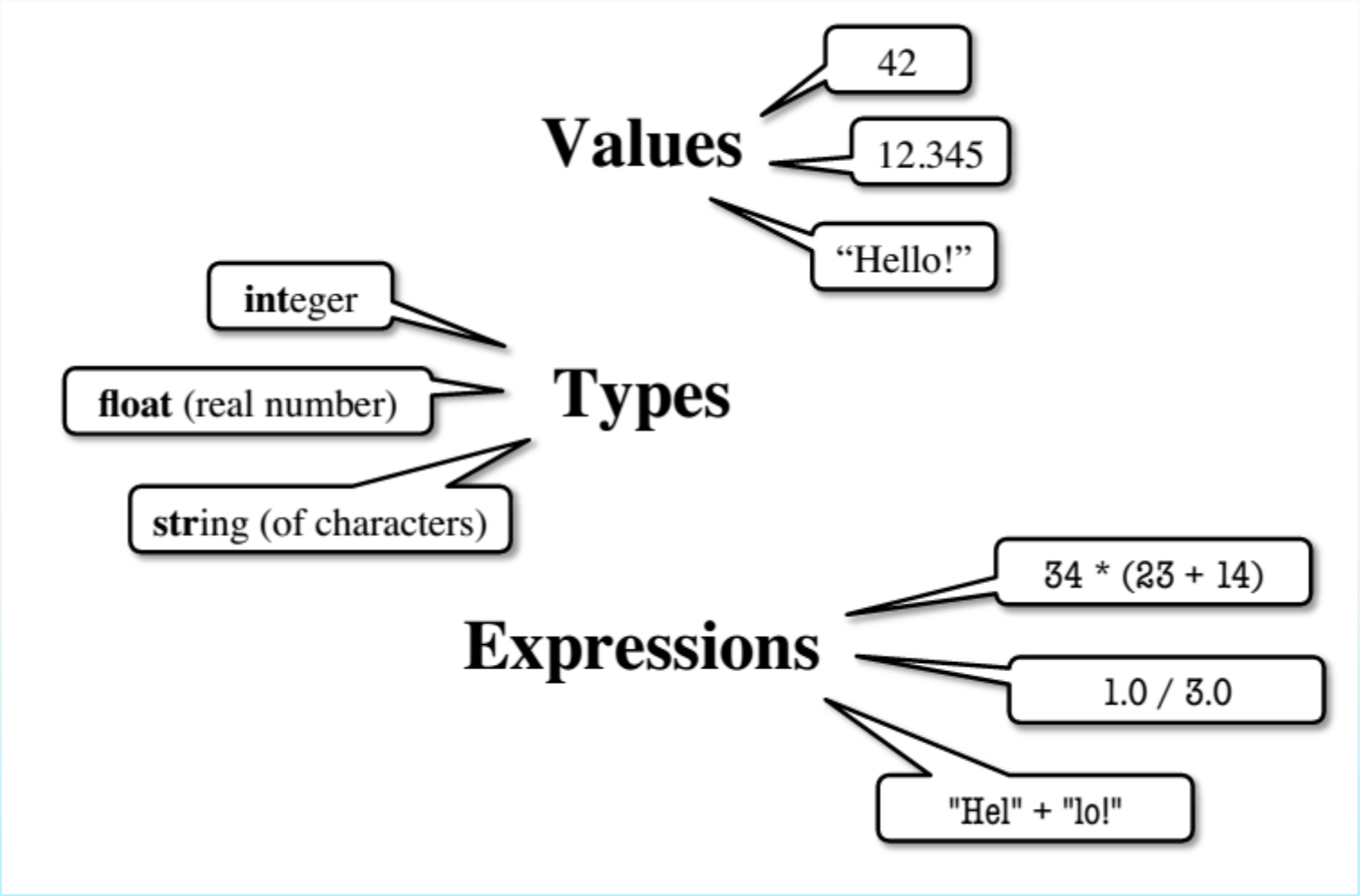
- Execute it

```
tcd@ofctl:~/python$ python hello.py
Spam: Hi Eggs, My name is Spam.
Eggs: Hello Spam!
3
1048576
```

Eggs

- Prerequisites
- Hello Spam!
- Data Types
- Variables, Operators, and Value Assignment
- Dynamic Typing
- String, List, Dictionary, Tuple
- Statements
- Odds-and-Ends
- Python Scoping and Function
- Python Module
- OOP, Class, Object

Components


Values
42
12.345
“Hello!”
Types
integer
float (real number)
string (of characters)
Expressions
34 * (23 + 14)
1.0 / 3.0
"Hel" + "lo!"

Type: int

- Values: -100, 0, 123, 1048576, etc.
- Operations: +, -, *, /, **(power), %(modulo)
- Operations on **int** values yield an **int**
  - 1 / 2 = 0
  - 3 / 2 = 1
- To obtain 1.5 for 3 / 2 ?!
  - float(3) / 2
  - 3.0 / 2
  - 3. / 2

Type: float

- Values: -10.23, 0.0, 123.456, etc.
- Operations: +, -, *, /, **, %
- Exponent notation:
  - A day has 8.64e4 (86400) seconds
  - Boltzmann constant: 1.38e-23 joule/K

Type: str

- Values: sequence of characters
  - ‘Hello Spam!’
  - ‘a # b & abc $ abcdefg’
- Operation: + for catenation
  - ‘He’ + ‘llo’ = ‘Hello’
  - ‘join’ + 4 + ‘free’ results in an error message
  - ‘join’ + ‘4’ + ‘free’ = ‘join4free’
- Operation: * for repeating
  - ‘TANET’ * 3 = ‘TANETTANETTANET’

Type: bool

- Values: True, False
- Operations: and, or, not, ^ (xor; exclusive or)
- Often come from comparisons:
  - Order comparison: <, <=, >, >=
  - Equality: ==, !=

Type Query and Conversion

- >>> type(2.5)
  - <type 'float'>
- >>> type(True)
  - <type 'bool'>
- >>> int(3.14)
  - 3
- >>> float(2)
  - 2.0

Eggs

- Prerequisites
- Hello Spam!
- Data Types
- Variables, Operators, and Value Assignment
- Dynamic Typing
- String, List, Dictionary, Tuple
- Statements
- Odds-and-Ends
- Python Scoping and Function
- Python Module
- OOP, Class, Object

Precedence of Operators

- Exponentiation: **
- Unary operators: +, -
- Binary arithmetic: *, /, %
- Binary arithmetic: +, -
- Comparisons: >, >=, <, <=
- Equality: ==, !=
- Logical not
- Logical and
- Logical or

Enhanced & Bitwise Operators

- Enhanced operators: +=, -=, *=, /=, %=, **=
  - x += y means x = x + y
  - x **= y means x = x ** y
  - No x++ expression in Python!
- Bitwise operators: &(and), |(or), ^(xor), ~(not), <<(left shift), >>(right shift)
  - `>>> x = 4`
  - `>>> x << 1`
  - `>>> x | 1`
  - `>>> x & 1`

Variables & Value Assignment

- >>> x = 3
- >>> x
  - 3
- >>> x = x+2
- >>> x
  - 5
- >>> x = x*2 + 2
- >>> x
  - 12

Eggs

- Prerequisites
- Hello Spam!
- Data Types
- Variables, Operators, and Value Assignment
- Dynamic Typing
- String, List, Dictionary, Tuple
- Statements
- Odds-and-Ends
- Python Scoping and Function
- Python Module
- OOP, Class, Object

Dynamic Typing

- Python is a dynamically typed language
  - Variables can hold values of any type
  - Variables can hold different types at different times
  - Use type(x)to find out the type of the value in x
  - Use names of types for conversion, comparison
- >>> x = 5
- >>> type(x)
  - <type 'int'>
- >>>x = x / 2.
- >>> type(x)
  - <type 'float'>

Type Confirmation

- >>> x = 5
- >>> type(x)
  - <type 'int'>
- >>> type(x) == float
  - False
- >>> type(‘abcd’) == str
  - True

Eggs

- Prerequisites
- Hello Spam!
- Data Types
- Variables, Operators, and Value Assignment
- Dynamic Typing
- String, List, Dictionary, Tuple
- Statements
- Odds-and-Ends
- Python Scoping and Function
- Python Module
- OOP, Class, Object

String: Text-Valued Variable

- Use ‘ (preferred) or “” to quote characters.
- When quotation marks are needed in the string, use another one.
  - ‘Happy “B”irthday’
  - “Alles ‘G’ute zum Geburtstag”
- Escape characters:

Char	Meaning
\'	single quote
\"	double quote
\n	new line
\t	tab
\\	backslash

Indices of a String

- >>> x = 'Robin van Persie'
- >>> x[0]
  - R
- >>> len(x)
  - 16
- >>> x[6:9]
  - 'van'
- >>> x[6:]
  - 'van Persie'

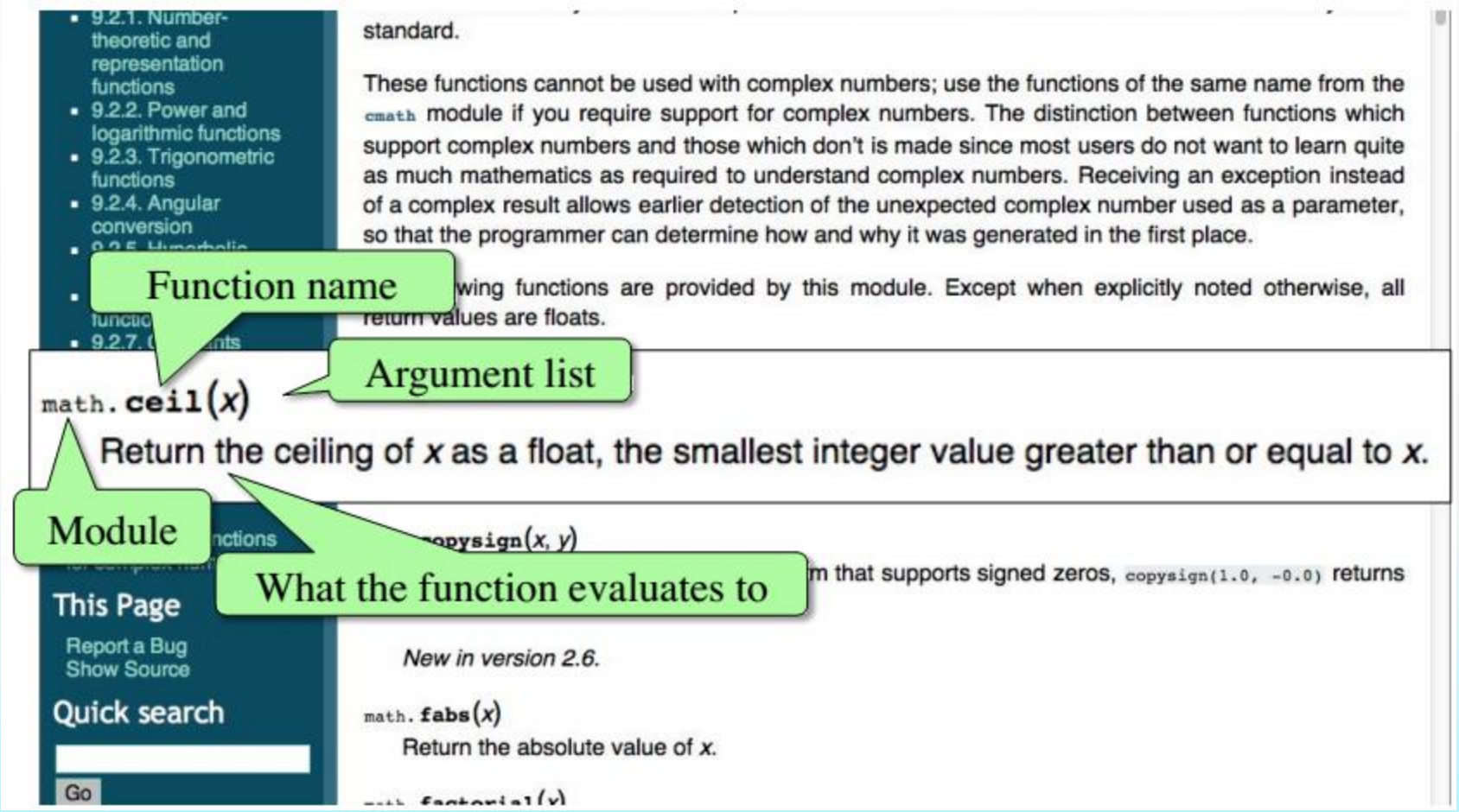
Other String Methods

- >>> 'x' in x
  - False
- >>> 'Per' in x
  - True
- >>> x.index('v')
  - 6
- >>> x.count('i')
  - 2

Even More String Methods?

- >>> from string import *
- >>> dir()
  - ['Formatter', 'Template', '__builtins__', '__doc__', '__name__', '__package__', 'ascii_letters', 'ascii_lowercase', 'ascii_uppercase', 'atof', 'atof_error', 'atoi', 'atoi_error', 'atol', 'atol_error', 'capitalize', 'capwords', 'center', 'count', 'digits', 'expandtabs', 'find', 'hexdigits', 'index', 'index_error', 'join', 'joinfields', 'letters', 'ljust', 'lower', 'lowercase', 'lstrip', 'maketrans', 'octdigits', 'printable', 'punctuation', 'replace', 'rfind', 'rindex', 'rjust', 'rsplit', 'rstrip', 'split', 'splitfields', 'strip', 'swapcase', 'translate', 'upper', 'uppercase', 'whitespace', 'zfill']
- Refer to https://docs.python.org/2/library/

Python Modules



Python Modules

- Frequently used modules:
  - io: Read/write from files
  - math: Mathematical functions
  - random: Generate random numbers
    - Distribution selectable
  - string: Useful string functions
  - sys: Information about your OS
- Refer to:
  - https://docs.python.org/2/library/
  - https://docs.python.org/2/py-modindex.html

Import a Python Module

- >>> import math
- >>> import random
- >>> math.pi
  - 3.141592653589793
- >>> print math.sin(math.pi/6)
- >>> random.random()
- >>> int(random.random()*10)+1
- >>> random.randint(1,10)

List

- A list is a series of data inside [ ](brackets).
- Items are separated by “,”(comma).
- Items are not necessarily of the same type.
- Items are editable.
- A list can contain another list

- >>> x = [2,5,'abc',10.1,'!!',True]
- >>> type(x[2])
  - <type 'str'>

List

- >>> type(x)
  - <type 'list'>
- >>> x[2:5]
  - ['abc', 10.1, '!!']
- >>> x[1] = ‘hello’
- >>> x
  - [2, 'hello', 'abc', 10.1, '!!', True]
- >>> x[2]=[4,5]
  - [2, 'hello', [4, 5], 10.1, '!!', True]

List

- >>> x.append(55)
- >>> x
  - [2, 'hello', [4, 5], 10.1, '!!', True, 55]
- >>> y = [2,8,5,7,1,4]
- >>> y.sort()
  - [1, 2, 4, 5, 7, 8]
- >>> y.pop()
- >>> y
  - [2, 4, 5, 7, 8]

List

- >>> y.reverse()
- >>> y
  - [8, 7, 5, 4, 2]
- >>> y.insert(2,'hi')
- >>> y
  - [8, 7, 'hi', 5, 4, 2]
- >>> del y[3]
- >>> y
  - [8, 7, 'hi', 4, 2]

List

- >>> y+[6,5,4]
  - [8, 7, 'hi', 4, 2, 6, 5, 4]
- >>> y
  - [8, 7, 'hi', 4, 2]
- >>> y += [6,5,4]
- >>> y
  - [8, 7, 'hi', 4, 2, 6, 5, 4]
- >> matrix = [[1,2,3],[4,5,6],[7,8,9]]
- >> matrix[2][1]

List

- >>> z = [6, 5, 0, 1, 2]
- >>> sum(z)
- >>> [min(z), max(z)]
- >>> range(5)
  - [0, 1, 2, 3, 4]
- >>> range(2,7)
  - [2, 3, 4, 5, 6]
- >>> range(2,7,2)
  - [2, 4, 6]

Dictionary

- >>> dic = {‘Robben’:11, ‘Sneijder’:10, ‘Kuyt’:15}
- >>> dic[‘Sneijder’]
  - 10
- >>> ‘Robben’ in dic
  - True
- >>> dic.keys()
  - ['Robben', 'Sneijder', 'Kuyt']
- >>> dic['Kuyt'] = ['goal','miss']

Dictionary

- >>> dic
  - {'Robben': 11, 'Sneijder': 10, 'Kuyt': ['goal', 'miss']}
- >>> dic['Kuyt'][0]
  - ‘goal’
- >>> dic.pop('Sneijder')
  - 10
- >>> dic
  - {'Robben': 11, 'Kuyt': ['goal', 'miss']}

Tuple

- A tuple is a series of data inside ()(parentheses).
- Items are separated by “,”(comma).
- Items are not necessarily of the same type.
- Items are NOT editable.
- A tuple can contain another list or tuple.

- >>> t1 = (6,5,4)
- >>> t2 = 9,8,7

Tuple

- >>> t1 + t2
  - (6, 5, 4, 9, 8, 7)
- >>> t2[1]
- >>> t3 = 'hello','hola','boujour'
- >>> t3 * 2
  - ('hello', 'hola', 'boujour', 'hello', 'hola', 'boujour‘)
- >>> t4 = t1[0],t2[1:3]
- >>> t4
  - (6, (8, 7))

Tuple

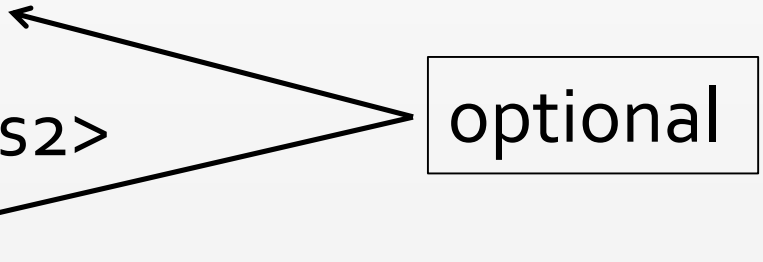
- >>> r = [x*5 for x in t1]
- >>> r
- >>> t1[0] = 1
  - TypeError: 'tuple' object does not support item assignment
- >>> t1 = list(t1)
- >>> t1[0] = 1
- >>> t1 = tuple(t1)
- >>> t1

Eggs

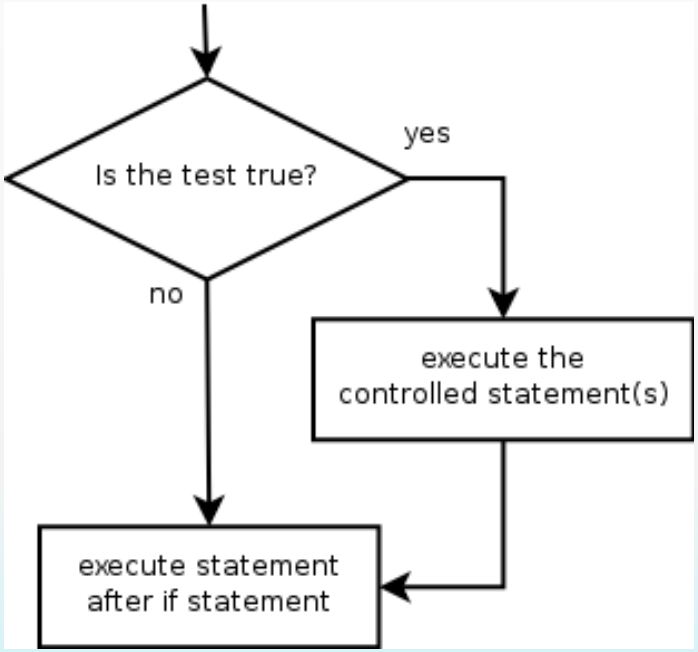
- Prerequisites
- Hello Spam!
- Data Types
- Variables, Operators, and Value Assignment
- Dynamic Typing
- String, List, Dictionary, Tuple
- Statements
- Odds-and-Ends
- Python Scoping and Function
- Python Module
- OOP, Class, Object

if/elif/else Statement

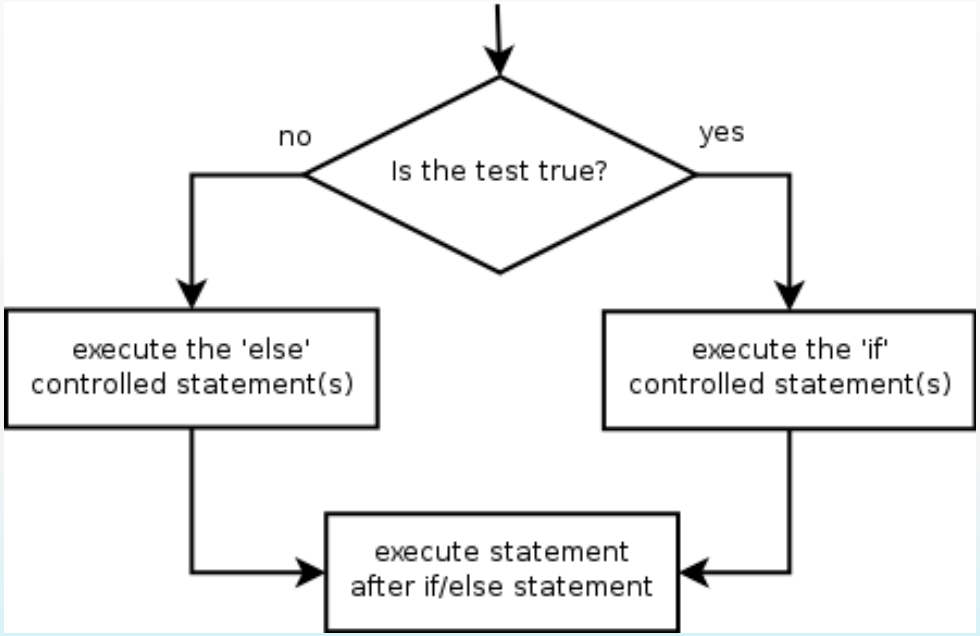
- Usage: if <test1>:
  <statements1>
  elif <test2>:
  <statements2>
  .
  .
  else:
  <statements3>



if/elif/else Statement

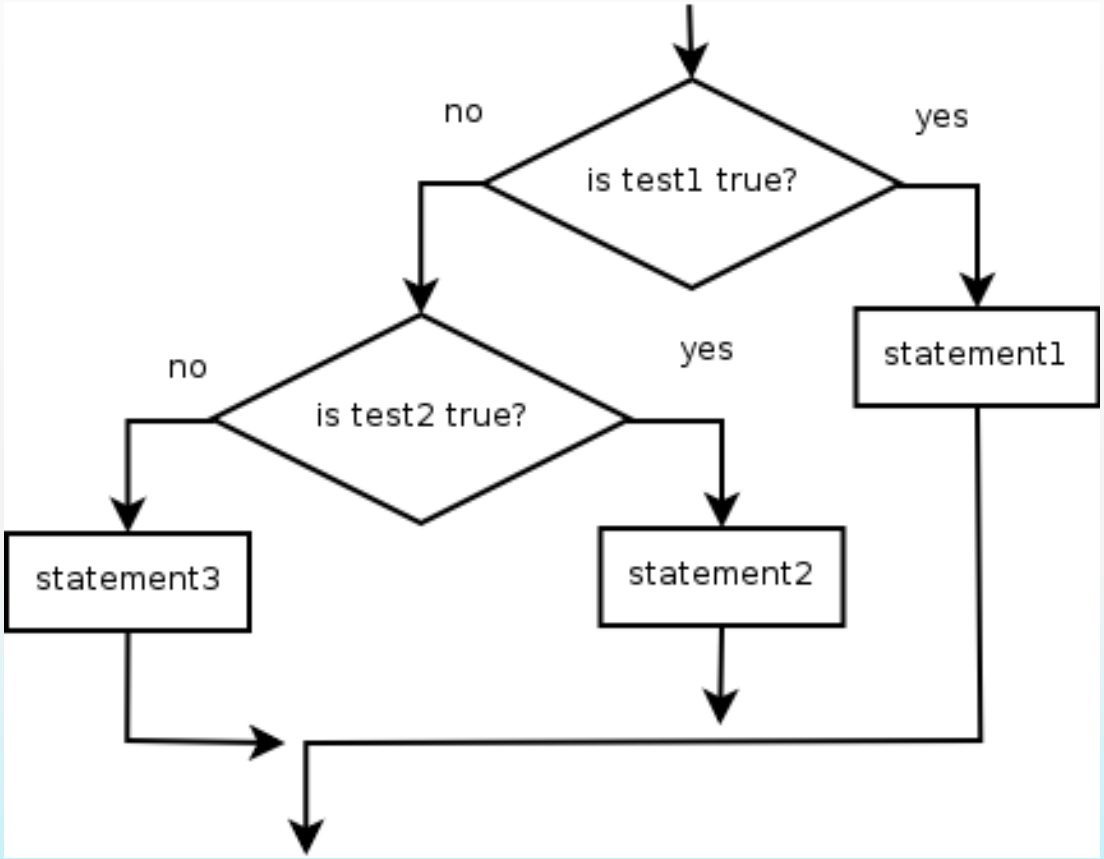


if



if/else

if/elif/else Statement



if/elif/else

if/elif/else Statement

- Example:

```
 1
 2 x = raw_input('enter a positive integer: ')
 3
 4 if not x.isdigit():
 5     print x,'is not a positive integer'
 6 elif int(x) < 10:
 7     print 'the integer is less than 10'
 8 else:
 9     print 'the integer is larger than 10'
10
```

map Function

- Usage: map(<function>,<sequence>)
- Example 1:
  - >>> x = [1.2, 3.4, 5.6, 7.8]
  - >>> y = map(round,x)
- Example 2:
  - >>> x = [‘55’, ‘66’, ‘derdi’, ‘y’]
  - >>> y = map(len,x)
- Example 3:
  - >>> x = [‘Abian’, ‘MyAngel’, ‘DongYingBaZhu’]
  - >>> y = (str.lower,x)

for Statement

- Usage: **for** <*variable*> **in** <*sequence*>:

  <statements>
- Example 1:
  - >>> for i in range(5):
  - … print i+1
- Example 2:
  - >>> for i in [2,4,6,8,10]:
  - … print '*' * i

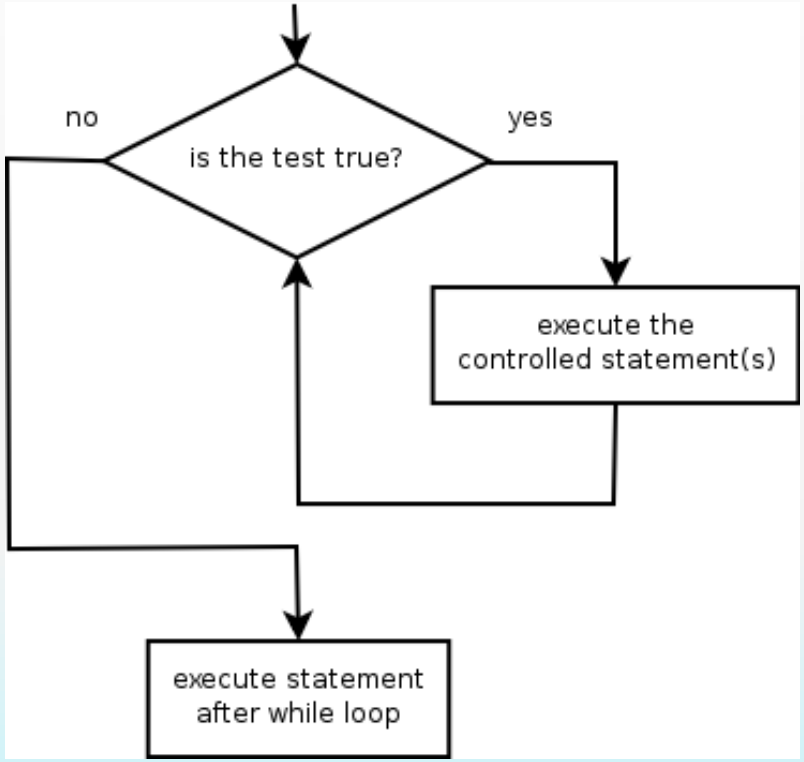
for Statement

- Example 3:
  - >>> t = [(1,2),(3,4),(5,6)]
  - >>> for (a, b) in t:
  - … print a*b
- Example 4:
  - >>> s = 'OnsOranje'
  - >>> for i in range(len(s)):
  - … print ' ' * i, s[i]

while Statement

- Usage: while <test>:
  <statements1>
  else:
  <statements2>

optional



while Statement

- Example 1:
  - >>> x = 'Spam'
  - >>> while x:
  - ... print x,
  - ... x = x[1:]
- Example 2:
  - >>> a, b = 0, 10
  - >>> while a < b:
  - ... print a,
  - ... a += 1

break/continue/pass

- Appears only in loops.
- **break**: exit the loop right away, neglecting all the statements in loop.
- **continue**: go to the beginning of the loop, neglecting the rest statements.
- **pass**: do nothing, ~~used to show off your specialty~~, used to occupy a position for further revision.

A Toy

```
 1
 2 import string
 3 nums = string.digits
 4
 5 while True:
 6     reply = raw_input('enter: ')
 7     if reply == 'stop':
 8         break
 9     else:
10         k = 0
11         for i in reply:
12             if i not in nums:
13                 k = 1
14                 break
15         if k == 0:
16             print int(reply) ** 2
17         else:
18             print reply.upper()
19 print 'au revoir!'
20
```

A “Larger” Toy - Craps

```
 1
 2 from random import randint
 3
 4 W = [7, 11]
 5 L = [2, 3, 12]
 6
 7 def dice():
 8     return randint(1,6) + randint(1,6)
 9
10 P = dice()
11 print 'rolled',P,'\t'
12 if P in W:
13     S = 'win'
14 elif P in L:
15     S = 'los'
16 else:
17     S = 'con'
18     pt = P
19     print 'roll',pt,'again to win'
```

```
20
21 while S == 'con':
22     P = dice()
23     print 'rolled',P,'\t'
24     if P == pt:
25         S = 'win'
26         break
27     elif P == 7:
28         S = 'los'
29         break
30
31 if S == 'win':
32     print 'win!'
33 elif S == 'los':
34     print 'lose!'
```

Eggs

- Prerequisites
- Hello Spam!
- Data Types
- Variables, Operators, and Value Assignment
- Dynamic Typing
- String, List, Dictionary, Tuple
- Statements
- Odds-and-Ends
- Python Scoping and Function
- Python Module
- OOP, Class, Object

Read and write file

- >>> myfile = open('filename','r')
- >>> x = myfile.readline()
- >>> y = myfile.read()
- >>> myfile.close()
- >>> newfile = open('towrite','w')
- >>> newfile.write('911 was a \n')
- >>> newfile.write('god-damned hoax!\n')
- >>> newfile.close()

Exception handling

- >>> x = dir(__builtins__)
- >>> y = []
- >>> for i in range(len(x)):
- ... if ‘Error’ in x[i]:
- ... y.append(x[i])
- >>> y
  - ['ArithmeticError', 'AssertionError', 'AttributeError', 'BufferError', 'EOFError', 'EnvironmentError', 'FloatingPointError', 'IOError', 'ImportError', 'IndentationError', 'IndexError', 'KeyError', 'LookupError', 'MemoryError', 'NameError', 'NotImplementedError', 'OSError', 'OverflowError', 'ReferenceError', 'RuntimeError', 'StandardError', 'SyntaxError', 'SystemError', 'TabError', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'ValueError', 'ZeroDivisionError']

try… except …

- Usage: try:

  <statement1>

  except (<error1>):

  <statement2>

  ……

  (else:)

- When an exception is encountered in statement1, jump to the corresponding block.
- **else** block will only be executed when no exception.

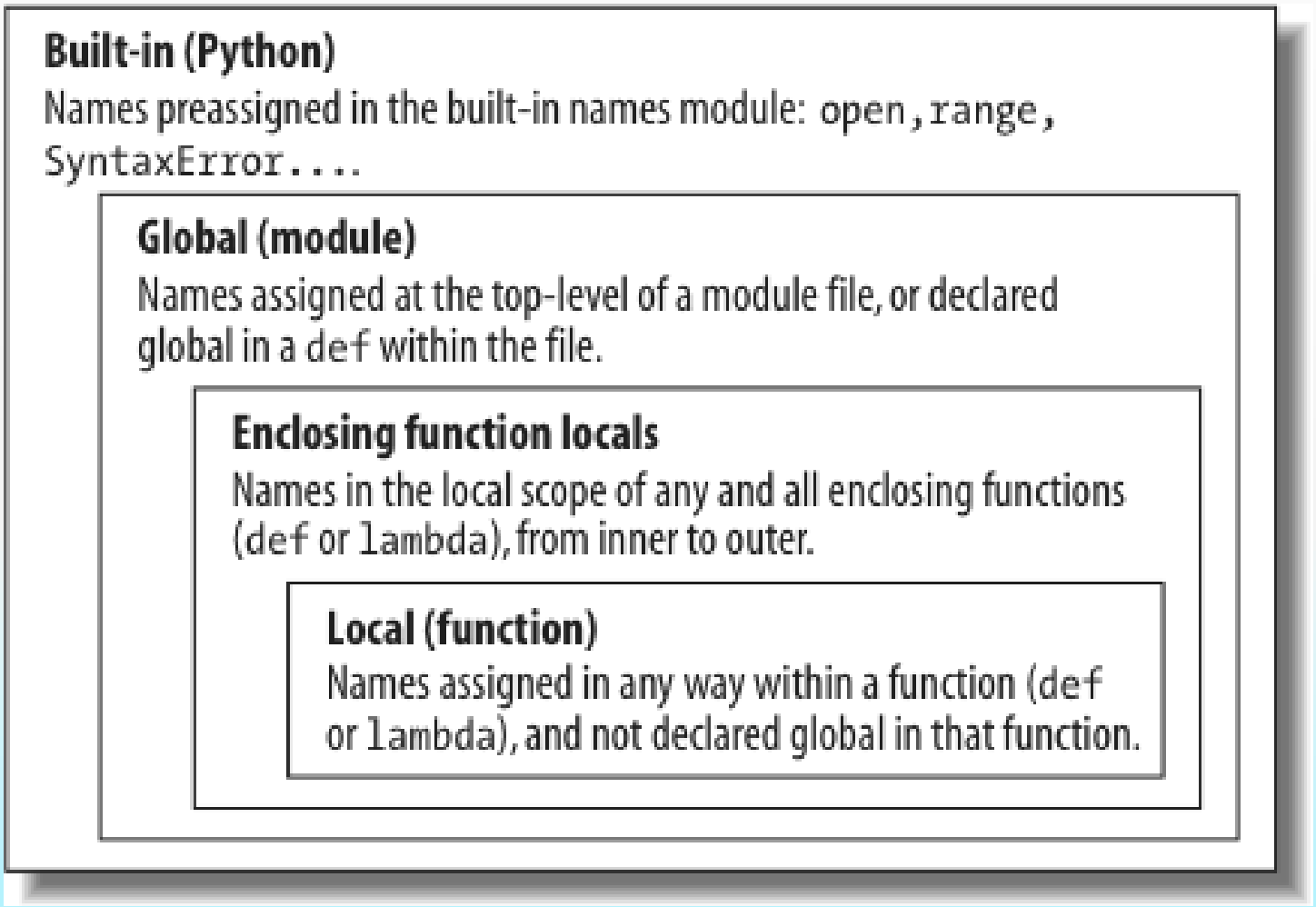
An Example

```
 1
 2 def fetcher(x, index):
 3     return x[index]
 4
 5 x = 'Schweinsteiger'
 6 y = 'Klose'
 7 a = 5
 8
 9 print x
10 print y
11
12 try:
13     print fetcher(x,a)
14 except IndexError:
15     print 'x[',a,'] does not exist'
16
17 try:
18     print fetcher(y,a)
19 except IndexError:
20     print 'y[',a,'] does not exist'
21
```

Eggs

- Prerequisites
- Hello Spam!
- Data Types
- Variables, Operators, and Value Assignment
- Dynamic Typing
- String, List, Dictionary, Tuple
- Statements
- Odds-and-Ends
- Python Scoping and Function
- Python Module
- OOP, Class, Object

Python Scoping



Functions

- tcd@ofctl:~/python$ vim simplearith.py

```
1
2 def add(a,b):
3     print a, '+', b, '=', a+b
4     return a+b
5
6 def minus(a,b):
7     print a, '-', b, '=', a-b
8     return a-b
9
```

- >>> import simplearith
- >>> x = simplearith.add(4,1)
  - 4 + 1 = 5
- >>> y = simplearith.minus(4,1)
  - 4 - 1 = 3

Functions

- Keywords: **def**, **return**
- Another example: Fibonacci sequence

```
 1
 2 def fibonacci(x):
 3     a,b = 1,1
 4     for i in range(x-1):
 5         a,b = b, a+b
 6     return a
 7
 8 fib = []
 9
10 for i in range(1,21):
11     fib.append(fibonacci(i))
12
13 print fib
14
```

Functions

- Execution time function definition is allowed.
- Function overloading is allowed.

```
 1
 2 x = input('enter an integer: ')
 3 y = raw_input('enter a string: ')
 4
 5 if x % 2 == 0:
 6     def func(z):
 7         return z[1::2]
 8 else:
 9     def func(z):
10         return z[0::2]
11
12 print func(y)
13
```

Functions

- Roll a die many times to see if it is fair.

```
 1 from random import randint
 2
 3 t = raw_input('times to roll: ')
 4 x = [0,0,0,0,0,0]
 5
 6 if not t.isdigit():
 7     print 'Antler is the hair in the ear of deer?!'
 8
 9 else:
10     for i in range(int(t)):
11         r = randint(1,6)
12         x[r-1] += 1
13
14     ctr = 0
15     while ctr < 6:
16         print ctr+1, 'point: ', x[ctr], 'times'
17         ctr += 1
18
```

- Of course you have to enter an integer!

More about Functions

- Default value setting:
  - >>> def fun1(a, b, c): print a,b,c
  - >>> def fun2(a,b,c=1): print a,b,c
- *pargs & **kargs are used to pass a variable number of arguments to a function.
  - *pargs is used to pass a non-keyworded, variable-length argument list.
  - **kargs is used to pass a keyworded, variable-length argument list.

More about Functions

```
 1
 2 def test_var_pargs(farg, *pargs):
 3     print 'formal arg:', farg
 4     for arg in pargs:
 5         print 'another arg:', arg
 6
 7 def test_var_kargs(farg, **kargs):
 8     print 'formal arg:', farg
 9     for key in kargs:
10         print 'another keyword arg: %s: %s' % (key, kargs[key])
11
12 def test_var_args_call(arg1, arg2, arg3):
13     print "arg1:", arg1
14     print "arg2:", arg2
15     print "arg3:", arg3
```

```
16
17
18 test_var_pargs(1, 'two', 3)
19 test_var_kargs(farg=1, myarg2='two', myarg3=3)
20
21 pargs = ('two', 3)
22 test_var_args_call(1, *pargs)
23 kargs = {'arg3': 3, 'arg2': 'two'}
24 test_var_args_call(1, **kargs)
25
```

Eggs

- Prerequisites
- Hello Spam!
- Data Types
- Variables, Operators, and Value Assignment
- Dynamic Typing
- String, List, Dictionary, Tuple
- Statements
- Odds-and-Ends
- Python Scoping and Function
- Python Module
- OOP, Class, Object

Haunted by Module

- A module is intrinsically a file containing Python definitions and statements.
- A module can contain executable statements as well as function definitions.
- The statements in a module are intended to initialize the module. They are executed only the first time the module name is encountered in an **import** statement.

Haunted by Module

- The built-in function **dir()** is used to find out which names a module defines. It returns a sorted list of strings.

- >>> dir()
- >>> dir(__bulitins__)
- >>> import math
- >>> dir(math)

reload

- Re-import an imported module after revision.
- Often used when restarting the entire application is costly.

- >>> import reloadeg
- >>> reloadeg.printer()
- >>> reload(reloadeg)
- >>> reloadeg.printer()

```
1
2 message = 'first version'
3
4 def printer():
5     print message
6
```

```
1
2 message = 'second version'
3
4 def printer():
5     print message
6
```

Import Nested Modules

- To figure out your current search path:
  - >>> import sys
  - >>> sys.path
    - [..... '/usr/local/lib/python2.7/dist-packages',.....]
- To import the module in dir1/dir2/, simply type:
  - import dir1.dir2.<module name>
- Remember to add a __init__.py in a user-defined module dir.

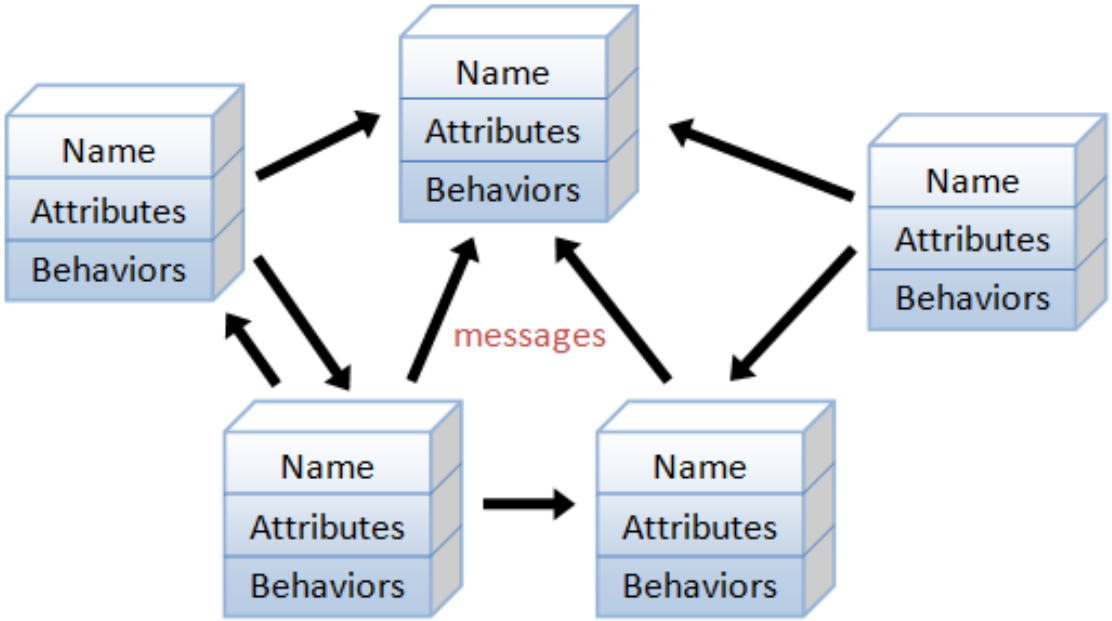
```
tcd@ofctl:~$ ls /usr/local/lib/python2.7/dist-packages/ryu
app     cfg.pyc  controller     flags.py   hooks.pyc     lib      ofproto   topology
base    cmd      exception.py    flags.pyc  __init__.py   log.py   services  utils.py
cfg.py  contrib  exception.pyc  hooks.py    __init__.pyc  log.pyc  tests     utils.pyc
```

Eggs

- Prerequisites
- Hello Spam!
- Data Types
- Variables, Operators, and Value Assignment
- Dynamic Typing
- String, List, Dictionary, Tuple
- Statements
- Odds-and-Ends
- Python Scoping and Function
- Python Module
- OOP, Class, Object

Viva OOP!

- Object-oriented programming is an approach to design modular, reusable systems.



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

Viva OOP!

- The goals of OOP are:
  - Increased understanding
  - Ease of maintenance
  - Ease of evolution

- Object is an instance of class.
- Class is the definition of object, defining the components and functionality.

An Example of Class

```
 1
 2 class firstcla:
 3
 4     def setData(self, value):
 5         self.data = value
 6     def display(self):
 7         print self.data
 8
 9     a = 1
10     b = 2
11
12
13 class secondcla(firstcla):
14
15     def display(self):
16         print 'current value = \'%s\'' % self.data
17     def dispsh(z):
18         print '*' * z
19
20     b = 3
21     c = 4
22
```

An Example of Class

- Object creation:
  - >>> x = firstcla()
  - >>> y = secondcla()
- Definition (methods & variables) visualization:
  - >>> dir(y)
    - ['__doc__', '__module__', 'a', 'b', 'c', 'data', 'display', 'dispsh', 'setData']
  - >>> dir(x)
    - ['__doc__', '__module__', 'a', 'b', 'data', 'display', 'setData']

Inheritance

- Inheritance is when an object or class is based on another object or class, using the same implementation or specifying implementation to maintain the same behavior.
- Usage: class derivedClass(baseClass1,…):

  <statements>
  - Depth-first
  - Left to right
  - Self -> Father -> Grandfather

An Example of Class

- Method (function) call:
  - >>> x.setData('Nigel de Jong')
  - >>> x.display()
  - >>> y.dispsh(5)
- Attribute reference:
  - >>> y.c ← Quite straightforward
  - >>> y.b ← 2 or 3 ??
  - >>> y.a ← Inherited attribute

Class Example Enhanced

- Initiation operation __init__ : things to do when right after the object is created.
- If you are familiar with C++, recall "constructor."

```
 1
 2 class firstcla:
 3
 4     def __init__(self):
 5         self.data = 'data unset!'
 6         self.x = 2048
 7     def setData(self, value):
 8         self.data = value
 9     def display(self):
10         print self.data
11
12     a = 1
13     b = 2
14
```

Set Values when Hatching

```
1
2 class firstcla:
3
4     def __init__(self, value='data unset!'):
5         self.data = value
6         self.x = 2048
7     def setData(self, value):
8         self.data = value
9     def display(self):
10        print self.data
11
```

- >>> x = firstcla()
- >>> y = firstcla('BRT?BUS?')
- >>> x.display()
- >>> y.display()