Programming with Python

Basic Topics

Mosky

Mosky:

- The examples and the PDF version are available at:
  - j.mp/mosky-programming-with-python.
- It is welcome to give me any advice of this slide or ask me the answers of the challenges.
  - mosky.tw

Mosky

- Projects
  - MoSQL
    mosql.mosky.tw
  - Clime
    clime.mosky.tw
  - Apt-Pool
    Apt-Add
    …
- Pinkoi staff
  pinkoi.com
- PyCon JP '12 Speaker
  pycon.jp
- PyCon TW '12 Speaker
  pycon.tw

Advertisement

- PyCon Taiwan 2013
  - pycon.tw
  - 5/25-26 @ Sinica
  - pythontw@googlegroups.com
- COSCUP 2013
  - coscup.org
  - 8/3-4 @ TICC
  - coscup-gereral@googlegroups.com

Topics

- Basic Topics
  - Python 2 or 3?
  - Environment
  - hello.py
  - Common Types
  - Flow Control
  - File I/O
  - Documentation
  - Scope
- Adv. Topics
  - Module and Package
  - Typing
  - Comprehension
  - Functional Technique
  - Object-oriented Prog.
  - Useful Libraries
- Final Project
  - A Blog System

An Investigation

Do you know _____ ?

- any other programming language
- Object-oriented
- Static Typing; Strong and Weak Typing
- Dynamic Typing
- Functor; Closure
- Functional Programming
- Web development

Python 2 or 3?

in short.

Python 2 or 3?

- Python 2.x
  - status quo
  - 2.7 is end-of-life release
  - harder for newcomers
  - more third-party lib.
  - 2to3.py
  - backported features:
    - What's News in Python 2.6
      docs.python.org/release/2.6.4/whatsnew/2.6.html
    - What's News in Python 2.7
      docs.python.org/dev/whatsnew/2.7.html
- Python 3.x
  - present & future
  - under active development
  - easier for newcomers
  - less third-party lib.
  - 3to2.py
  - new features:
    - What's News in Python 3.0
      docs.python.org/py3k/whatsnew/3.0.html

Python 2 or 3? (cont.)

- Use Python 3 if you can.
- Decide Python 2 or 3 by the library you will use.
- Today, we will go ahead with Python 2.

  And introduce you to the changes in Python3.

Environment

Is a python in your computer?

On Linux or Mac

- Python is *built-in* on Linux or Mac.
- All you have to do is check the version.
  Type "`python`" in any terminal.

```
Python 2.7.3 (default, Sep 26 2012, 21:51:14)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license"
for more information.
>>>
```

On Windows

- Download the installer from: "http://python.org/download"
- Install it.
- Add the Python's `PATH`.
  - Computer → System Properties → Advanced system settings → Advanced tab → Environment Variables → System Variables → find `PATH`.
  - `"...;C:\Python27"`

Editor / IDE

- The Editors
  - Sublime Text 2
    www.sublimetext.com
  - VIM
    wiki.python.org/moin/Vim
  - Gnome Text Editor (gedit)
  - Notepad++
    notepad-plus-plus.org
  - ...
- The IDE
  - IDLE
    - Debian-base:
      `sudo apt-get install idle`
    - Windows:
      Use the Start Menu to search "IDLE"
- The others:
  - wiki.python.org/moin/PythonEditors

The Python Shell

- Type "`python`" in terminal.
  - `>>>`
  - `...`
- Leaving a shell:
  - `exit()`
  - Linux or Mac: `Ctrl+D`
  - Windows: `Ctrl+Z<Enter>`

The python Command

- Enter Python shell without arguments.
- `python hello.py`
- `python -c 'print "Hello, World!"'`
- `python -m SimpleHTTPServer`

hello.py

Say hello to Python.

hello.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# file: hello.py

def hello(name=None):

    if name:\n
        return 'Hello, %s!' %
name
    else:
        return 'Hello, Python!'
```

- `#!` the shebang.
- `# -*-` defines the encoding of this file.
- `#` means the comments.
- `:` starts a block.
- Block uses 4-space indent.
- The statements ends with `\n`.

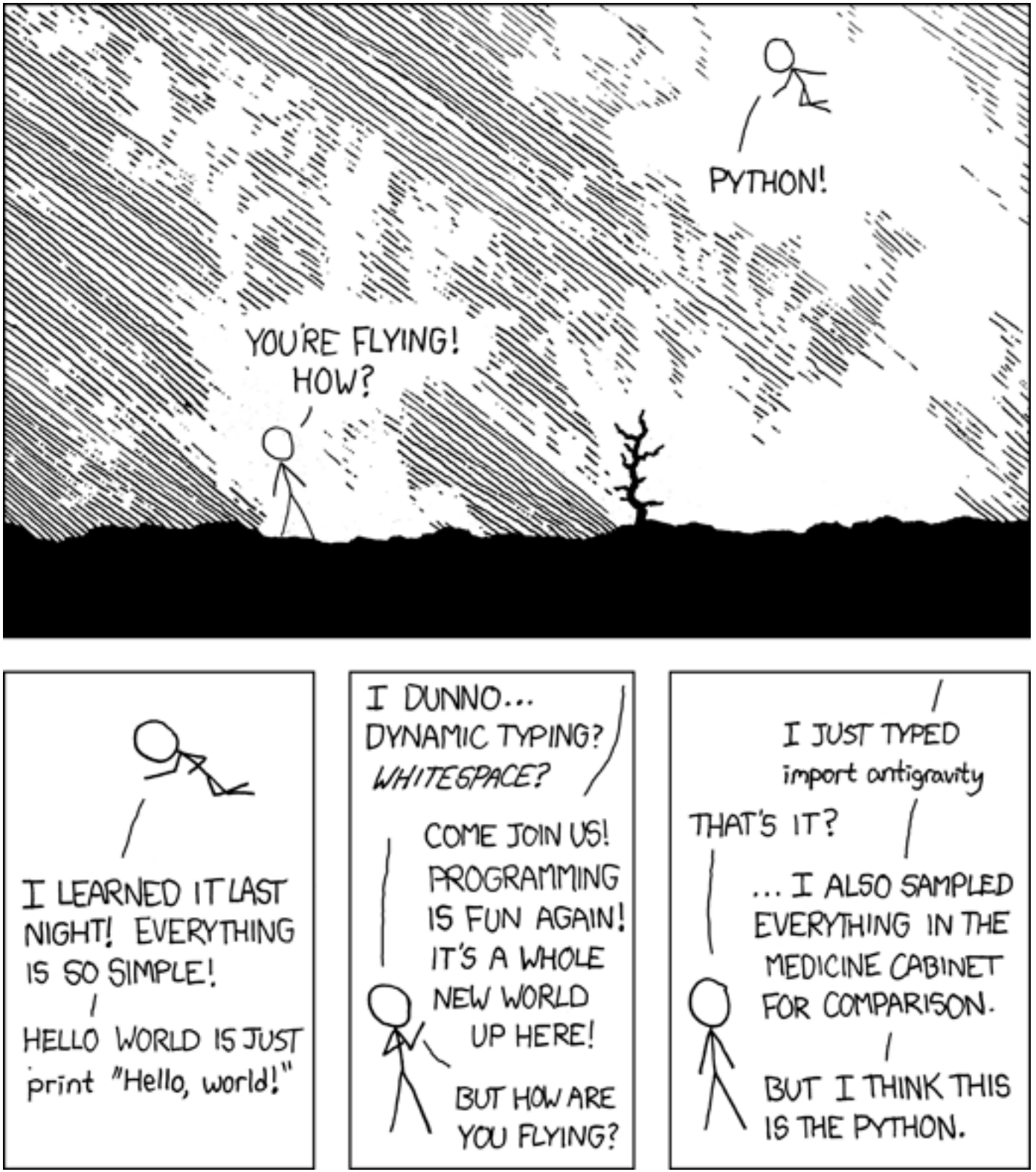
hello.py (cont.)

```
if __name__ == '__main__':
    import sys

    if len(sys.argv) >= 2:
        print hello(sys.argv[1])
    else:
        print hello()
```

- `__name__`, the name of module.
- `import` is important.

  The usage:
- `import sys`
- `from sys import argv`
- `… as alias`


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

The `print` Statement

```
print 'End with a new line char.'
print 'Print', 'multiple', 'strings.'
print 'End with a space.',
print # print a new line char
```

The `print` function in Python 3

```
print('End with a new line char.')
print('Print', 'multiple', 'strings.')
print('End with a space.', end=' ')
print() # print a new line char

print('End with a space.', end='')
print('a', 'b', 'c', seq=',')
```

Common Types

Without it we can do noting

Common Types

- Numeric
  - Integer `100`
  - Float `10.0`
  - `Long 100L`
  - Complex `1+1j`
  - Boolean `True, False`
- Sequence
  - String `""`
  - Unicode `u""`
  - List `[,]`
  - Tuple `(,)`

Common Types (cont.)

- Mapping
  - Dictionary `{:}`
- Set
  - Set `{,}`
  - Frozen Set `forzenset(...)`

Integer, Float and Long

- 3+3
- 3-3
- 3*3
- 3/3
- 6/2*(1+2)

  → int
  (as long in C)

- divmod(5, 2)

  → tuple (not numeric)
- 5/2

  → int (truncated)
- 5.0/2

  → float
  (as double in C)
- 5.0//2

  → float (floored)
- 2**1000

  → long (∞ precision)

Integer and Float in Python 3

- 3+3
- 3-3
- 3*3
- 3/3
- 6/2*(1+2)

  → int
  (∞ precision)

- divmod(5, 2)

  → tuple (not numeric)
- 5/2

  → float
- 5.0/2

  → float
  (as double in C)
- 5.0//2

  → float (floored)
- 2**1000

  → int (∞ precision)

Note: The Variables

- `x = 1`
- `x + 1`
  `→ 2`
- `y = 2`
- `x + y`
  `→ 3`
- `y += 3`
  `→ 5`

- `x < y`
  `→ True`

- `bin(y)`
  `→ '0b101'`
- `bin(y | 0b011)`
  `→ '0b111'`

A Trap of the Integer

- `weight = 49`
- `height = 163`
- `bmi = weight / (height / 100) ** 2`
- `bmi`

  `→ 49`

- `(height / 100)`

  `→ 1`

A Trap of the Integer (cont.)

- `weight = 49.0`
- `height = 163.0`
- `bmi = weight / (height / 100) ** 2`
- `bmi`

  → `18.442545824080696`

Complex

- `1j * 1J`
- `1j * complex(0,1)`
- `3 + 1j*3`
- `(3+1j)*3`
- `(1+2j)/(1+1j)`

  → complex

- `a = 3.0+4.0j`
- `float(a)`

  → TypeError

- `a.real`

  → 3.0

- `a.imag`

  → 4.0

- `abs(a)`

  `# = sqrt(a.real**2 + a.imag**2)`

  → 5

Boolean

- `not False`
- `True and True`
- `False or True`

- `False +1`
  `→ 1`
- `True +1`
  `→ 2`

Comparison:

- `10 < 100`
- `10 < 10.0`
- `10 <= 10.0`
- `10 == 10.0`
- `10 != 10.0`
- `x is y`

String and Unicode

'...' is equal to "..."

String (immutable seq.)

- `'中文'`
- `'嗨, \nPython !'`
- `r'嗨, \nPython !'`
- `''' ... '''`

Unicode (immutable seq.)

- `u'嗨, \nPython !'`
- `ur'嗨, \nPython !'`
- `u''' ... '''`

Functions

- `ord('A')`
- `chr(65)`
- `ord(u'中')`
- `unichr(20013)`; ~~`chr(20013)`~~

Decoding (String → Unicode)

- `'中文'.decode('utf-8')`
- `unicode('中文', 'utf-8')`

Encoding (Unicode → String)

- `u'中文'.encode('utf-8')`
- ~~`str(u'中文')`~~
- ~~`str(u'中文', 'utf-8')`~~

Bytes and String in Python 3

'...' is equal to "..."

Bytes (immutable seq.)

- ~~b'中文'~~
- b'嗨, \nPython ! '
- br'嗨, \nPython ! '
- b''' ... '''

String (immutable seq.)

- '嗨, \nPython ! '
- r'嗨, \nPython ! '
- ''' ... '''

Functions

- ord(b'A')
- chr(65)
- ord('中')
- ~~unichr(20013);~~ chr(20013)

Decoding (Bytes → String)

- b'中文'.decode('utf-8')
- str(b'中文', 'utf-8')

Encoding (String → Bytes)

- '中文'.encode('utf-8')
- ~~bytes('中文')~~
- bytes('中文', 'utf-8')

Unicode Does Matter!

- `b = '中文'`
- `len(b)`

  `→ 6`
- `len(b.decode('utf-8'))`

  `→ 2`

String and Unicode (cont.)

- They have a lot of methods:

  capitalize center count decode encode endswith expandtabs find rfind format index rindex isalnum isalpha isdigit islower isspace istitle isupper join ljust rjust lower partition rpartition replace split rsplit splitlines startswith rstrip strip lstrip swapcase title translate upper zfill

- ref: docs.python.org/2/library/stdtypes.html#string-methods

String and Unicode (cont.)

String formatting:

- % (modulo)
  - ref: docs.python.org/2/library/stdtypes.html#string-formatting-operations
- str.format
  - ref: docs.python.org/2/library/string.html#formatstrings

List and Tuple

List (mutable seq.)

- `[]`
- `['item']`
- `['s', 100, u'unicode']`
- `list('abc')`
- `'a b c'.split(' ')`
- `'\n'.join(['spam', 'eggs'])`
- `x, y = [1, 2]`
- `x, y = [y, x]`

Tuple (seq.)

- `tuple()`
- `('item', )`
- `('s', 100, u'unicode')`
- `tuple('abc')`
- `'\n'.join(('spam', 'eggs'))`
- `x, y = (1, 2)`
- `x, y = (y, x)`

Sequence

Sequence

- x in s # performance?
- x not in s
- s + t
- s * n, n * s
- s[i]
- s[i:j]
- s[i:j:k]
- len(s)
- s.index(x)
- s.count(x)

Mutable Seq.

- s[i] = x
- s[i:j] = t
- del s[i:j]
- s[i:j:k] = t
- s.append(x)
- s.insert(i, x)
- s.pop([i])
- s.remove(x) # performance?
- s.extend(t)
  in-place
- s.sort([cmp[, key[, reverse]]])
- s.sort([key[, reverse]]) # Py 3
- s.reverse()

Sequence Comparison

- `(0, 0, 0) < (0, 0, 1)`
- `[0, 0, 0] < [0, 0, 1]`
- `(0, ) < (0, 0)`
- `'ABC' < 'C' < 'Pascal' < 'Python'`
- `(1, 2, 3) == (1.0, 2.0, 3.0)`

- `'A' == 'A'`
- `'A' > 65`
- `'A' > 66`
- `('A', ) > (66, )`

Sequence Comparison in Python 3

- `(0, 0, 0) < (0, 0, 1)`
- `[0, 0, 0] < [0, 0, 1]`
- `(0, ) < (0, 0)`
- `'ABC' < 'C' < 'Pascal' < 'Python'`
- `(1, 2, 3) == (1.0, 2.0, 3.0)`

- `'A' == 'A'`
- `'A' > 65` → `TypeError`
- `'A' > 66` → `TypeError`
- `('A', ) > (66, )` → `TypeError`

Sequence (cont.)

Slicing and Slice object:

```
- s = range(10)
- t = s
- t[0] = 'A'
- print s
- t is s

- t = s[:]
- t is s
```

```
- s = 'I am a str.'
- s[:-3]

- s.reverse()
  → TypeError
- s[::-1]
- ''.join(reversed(s))

- slice(None, None, -1)
```

Mapping

Dict. (mutable map.)

- {}
- {'A ': 1, 'B': 2, 'C': 3}
- dict({...})
- dict(A=1, B=2, C=3)

- k = 'ABC'
- v = [1, 2, 3]
- pairs = zip(k, v)
- dict(pairs)

- len(d)
- d[k]
- d[k] = v
- del d[k]
- k in d, k not in d
- d.copy()
- d.get(key[, default])
- d.setdefault(key[, default])
- d.items(), d.keys(), d.values()
- d.pop(key[, default)
- d.update([other])

  …

Set

Set (mutable set)

- `set()`
- `{'A', 'B', 'C'} # Py3`
- `set('ABC')`
- `set(['A','B','C'])`

- `len(s)`
- `x in s, x not in s`
- `s.copy()`
- `s.add(elem)`
- `s.discard(elem)`
- `s.pop()`
- `s |= other`
- `s &= other`
- `s | other | ...`
- `s & other & ...`
- `s < | <= | == | > = | > other`

  …

Flow Control

in Python is grace and easy to learn.

The `if` Statement

```
if [condition 1]:
    …
elif [condition 2]:
    …
elif [condition 3]:
    …
else:
    …
```

[exp. if true] `if` [condition] `else` [exp. if false]

Truth Value Testing

They are same as `False` in a boolean context:

- None
- False
- Zeros (ex. `0, 0.0, 0L, 0j`)
- Empty containers (ex. `'', [], {}`)
- `__nonzero__()` or `__len__()` returns `0` or `False`

Truth Value Testing (cont.)

- `if None ...`
- `if [] ...`
- `if [0] ...`
- `if [[]] ...`
- `if "" ...`
- `if {} ...`
- `if {False:False} …`

…

The `for` Statement

```
for [item] in [iterable]:
    …
```

```
for i in [0, 1, 2]:
    print i
```

```
for i in range(3):
    print i
```

```
for i in xrange(3):
    print i
```

The `for` Statement in Python 3

```
for [item] in [iterable]:
    …
```

```
for i in [0, 1, 2]:
    print i
```

```
for i in range(3):
    print i
```

~~`for i in xrange(3):`~~
~~`    print i`~~

The for Statement (cont.)

```
for i in range(1, 3):
    print i
```

```
for i in range(3, -1, -1):
    print i
```

```
s = [1, 2, 3]
t = 'xyz'
for i, j in zip(s, t):
    print i, j
```

```
s = [...]
for i, item in enumerate(s):
    print i, item
```

The `for` Statement (cont.)

- It is like `for … each` in other language.
  - Note: Python hasn't other `for` loop.
- It can iterate all of *iterable* object.
  - In other words, the object which defined `__iter__`.
  - ex. sequence, mapping, set, ...

Challenge 1: A Pyramid

- Use `for` loop to build a pyramid on right.
  - without limit.
  - limit: in two lines
    - hint: string formatting

```
   *
  ***
 *****
*******
```

Challenge 2-1: Count the Chars

- Use `for` loop to count the sentence on right.
  - without limit.
  - limit: without if
    - hint: use get

```
"Please count the
characters here."
```

```
{'P': 1, ...}
```

Challenge 2-2: Collect the Chars

- Use `for` loop to collect the chars.
  - limit: use `setdefault`

```
"Here are UPPERCASE
and lowercase chars."
```

```
{'c': ['C', 'c',
'c'], ...}
```

The `while` Statement

```
tasks = [...]
while tasks:
    …
```

- It leaves the loop once the `tasks` is empty.

```
while 1:
    …
```

- A infinite loop.
- It is better to use block mechanism in a loop.
  - ex. I/O block

The `break, continue` Statement

```
loop …:
    if …: break
```

- It terminates a loop.

```
loop …:
    if …: continue
```

- It continues with the next iteration.

The `break, continue` Statement (cont.)

- They do the same thing in both C and Python.
- Using `break` or `continue` is *encouraged*.
  - take the place of the complicated condition in a `while`.
  - faster, because Python is interpreted.
- Just use them.

The `pass` Statement

- Do nothing.

The `else` Clause on Loops

```
loop …:
    …
else:
    …
```

- No a clause on the `if` statement!
- If the loop isn't broken by any `break` statement, the `else` block is executed.
- It replaces the flags we usually used.

Challenge 3-1: The Primes

- Try to filter the primes from [2, 100).
  - without limit.
  - limit: use loop's `else`

```
[2, 3, 5, 7, 11, 13,
17, 19, 23, 29, 31,
37, 41, 43, 47, 53,
59, 61, 67, 71, 73,
79, 83, 89, 97]
```

The `try` Statement

```
try:
    …
except LookupError, e:
    …
except (IndexError, KeyError), e:
    …
else:
    …
finally:
    …
```

The `try` Statement in Python 3

```
try:
    …
except LookupError as e:
    …
except (IndexError, KeyError) as e:
    …
else:
    …
finally:
    …
```

The `try` Statement (cont.)

- For avoiding to catch the exception we don't expect, you should:
  - reduce your code in `try` block.
    - move them to `else` block.
  - make the exception precise in `except` statement.
    - Avoid using `Exception`.
    - ref: docs.python.org/2/library/exceptions.html#exception-hierarchy
- Release the resource in `finally` block.
  - or use context manager
  - ex. file, socket, …
- `raise SomeError`

The def Statement

```
def f(x, y):
    return (x, y)
```

```
f(1, 2)
f(y=2, x=1)
f(*(1, 2))
f(**{'y': 2, 'x': 1})
```

```
def f(x, y=2):
    return (x, y)
```

```
f(1)
f(x=1)
f(*(1, ))
f(**{'x': 1})
```

The def Statement (cont.)

```
def f(*args):
    return args
```

```
f(1, 2, 3)
# f(y=2, x=1) # → TypeError
f(*(1, 2, 3, 4))
# f(**{'y': 2 ,'x': 1})
# → TypeError
```

```
def f(**kargs):
    return kargs
```

```
# f(1, 2) # → TypeError
f(x=1, y=2, z=3)
# f(*(1, 2)) # → TypeError
f(**{'x': 1, 'y': 2, 'z': 3})
```

The def Statement (cont.)

```
def f(x, *args):
    return x, args
```

```
f(1, 2, 3)
# f(y=2, x=1) # → TypeError
f(*(1, 2, 3, 4))
# f(**{'y': 2, 'x': 1})
# → TypeError
```

```
def f(x, **kargs):
    return kargs
```

```
# f(1, 2) # → TypeError
f(x=1, y=2, z=3)
# f(*(1, 2)) # → TypeError
f(**{'x': 1, 'y': 2, 'z': 3})
```

The def Statement (cont.)

```
def f(*args, y):
    return kargs
```

→ SyntaxError

```
def f(*args, **kargs):
    return args, kargs
```

```
f(1, 2, 3)
f(y=2, x=1)
f(*(1, 2, 3, 4))
f(**{'y': 2, 'x': 1})
```

The def Statement in Python 3

```
def f(*args, k):
    return kargs
```

```
F(1, 2, 3)
# f(x=1, k=2) # → TypeError
f(*(1, 2, 3, 4))
# f(**{'x': 1, 'k': 2})
# → TypeError
```

```
def f(*args, k, **kargs):
    return args, kargs
```

```
f(1, 2, 3)
f(x=1, k=2)
f(*(1, 2, 3, 4))
f(**{'x': 1, 'k': 2})
```

The `def` Statement (cont.)

```
def f(): pass
def g(): pass
d = {'x': f, 'y': g}
d['x']()
```

- Python functions are *first-class* functions.
  - It means you can pass functions as arguments, and assign functions to variables.
  - It is like the *function pointers* in C.

An Example of Using while, try and def.

```
# file: ex_try.py

def take_int(prompt='Give me a int: '):
    while 1:
        try:
            user_input = int(raw_input(prompt))
        except ValueError, e:
            print 'It is not a int!'
        else:
            return user_input

if __name__ == '__main__':
    x = take_int()
    print 'I got a int from user: %d' % x
```

```
$ python ex_try.py
Give me a int: str
It is not a int!
Give me a int: abc
It is not a int!
Give me a int: 100
I got a int from user:
100
$
```

A Trap of the Default Value

```
# file: ex_defval_trap.py

def f(items=[]):
    items.append(1)
    return items

if __name__ == '__main__':
    print f() # -> [1]
    print f() # -> [1, 1]
    print f() # -> [1, 1, 1]
```

- Because the list is created when the function is defined.
- Avoid to use the mutable types as the default value.

Challenge 4: A BMI Calculator

- BMI: Body Mass Index
  - BMI = weight (KG) ÷ height $(M)^2$
  - < 18.5 → Underweight
  - [18.5, 25) → Normal weight
  - [25, 30) → Overweight
  - >= 30 → Obesity
- Write a BMI calculator.
  - without limit.
  - limit: only one if
    - hint: use loop

```
Enter your height (M):
1.63
Enter your weight (KG):
49
---
Your BMI is:
18.44 (Underweight)
Ideal weight is between:
49.15 ~ 66.42
```

File I/O

Open anything with the `open`.

The `file` Object

```
f = open('input.txt')
print f.read()
f.seek(0)
for line in f:
    print line,
f.close()
```

```
f = open('output.txt', 'w')
f.write('a line.\n')
f.close()
```

The Context Manager

```
with open('input.txt') as f:
    for line in f:
        print line,
f.close()
```

- Python 2.5↑
  - Python 2.5.x: `from __future__ import with_statement`
  - Python 2.6↑: It is mandatory.

Challenge 2: Count the Chars (cont.)

- limit 3: with the files

```
The path of input:
input.txt
The path of output:
output.txt
---
The result was
written.
```

The csv Moudle

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# file: ex_csv.py

import csv

with open('ex_csv.csv') as f:
    for row in csv.reader(f):
        print row
```

```
1, apple
2, orange
3, watermelon
```

```
['1', ' apple']
['2', ' orange']
['3', ' watermelon']
```

The os.path Moudle

```
# file: ex_os_path.py
from os import walk
from os.path import join

def list_files(path):
    paths = []
    for root, dir_names, file_names in walk(path):
        for file_name in file_names:
            paths.append(join(root, file_name))
    return paths

if __name__ == '__main__':

    import sys
    from os.path import abspath, dirname

    if len(sys.argv) == 2:
        path = abspath(dirname(sys.argv[1]))
        for path in list_files(path):
            print path
    else:
        print 'It requires a path as argument.'
```

```
$ python ex_os_path.py
It requires a path as argument.
$ python ex_os_path.py .
…/1
…/b/4
…/a/2
…/a/3
```

Documentation

The documentation is everywhere.

The help Function

- In Python shell:
  - `help(open)`
  - `dir(open)`
  - `'\n'.join(dir(open))`
- In terminal:
  - `$ pydoc SimpleHTTPServer`
  - `$ pydoc csv`
  - `$ pydoc os.path`

Your Documentation

```
# file: ex_doc.py

'''module-level doc.'''

def f(x):
    '''A short sentence describes
this function.

    About the parameters, return
value or any other detail ...
    '''
    pass
```

```
$ pydoc ex_doc
Help on module ex_doc:

NAME
    ex_doc - module-level doc.

FILE
    /home/mosky/programming-with-python/ex_doc.py

FUNCTIONS
    f(x)
        A short sentence describes this
function.

        About the parameters, return value or
any other detail ...
```

Scope

Where is the x?

Function Scope

```
# file: ex_scope.py

x = 'global'

def f():
    if 1:
        x = 'local'
    return x

if __name__ == '__main__':
    print x
    print f()
```

```
$ python ex_scope.py
global
local
$
```

- Scopes are decided by *functions*.

The LEGB Rule

```
# file: ex_LEGB.py

global_var = 100

def f():
    enclosed_var = 10

    def g():
        local_var = 1
        return sum([local_var, enclosed_var,
global_var])

    return g()

if __name__ == '__main__':
    print f() # -> 111
```

- `return` …
  - Local (in function)
  - Enclosed
  - Global
  - Built-in

Challenge 3-2: The Primes (cont.)

- limit 1: Sieve of Eratosthenes.
- limit 2: use `set`.

```
[2, 3, 5, 7, 11, 13,
17, 19, 23, 29, 31,
37, 41, 43, 47, 53,
59, 61, 67, 71, 73,
79, 83, 89, 97]
```

Challenge 5: Mix All

- You have many functions now.
- Try to write a CLI program to trigger your functions.
  - without limit
  - limit: without `if`.

```
$ python mix.py pyramid 10
…
$ python mix.py primes 100
…
$ python mix.py bmi 1.63 49
…
$ python mix.py blah blah
Please check your args.
```

Adv. Topics

There is another slide.