

Python

--- prepare for django

A Brief Introduction of Django

- A Lightweight Web Framework
- Compatible With Google App Engine
- ORM Supported
- An Open-source Project

Basic Skills

- Variables
- Computation
- Flow Controls
- Functions
- Module
- OOP

Variables

Types	Examples
int	-1,2,0xE8C6
float	1.25,4.3e+2
complex	2+2j,3-j
str	'abc','1234'
list	['abc','cde'],3,1.5]
tuple	(1,3),('12','2','4')
dictionary	{'abc':1,122:'dd'}
bool	True, False

Emphasis on 4 Sequencing Types

- Operations (append, delete ...)
- Slicing
- List Mapping
- ""
- Built-in Functions & Factory Functions

Operations (str)

- `>>> s = 'Django is cool'`
- `>>> words = s.split()`
- `>>> '::'.join(words)`
- `>>> s.upper()`
- `>>> s.title()`
- `>>> s.capitalize()`
- `>>> s.replace('Django','Python')`
- `>>> s.startswith('Django')`

Operations (list)

- `>>> book = ['Python','Develop',8] # creation`
- `>>> book.append(2008)`
- `>>> book.insert(1,'Web')`
- `>>> 'Django' in book`
- `>>> book.remove(8)`
- `>>> book.pop(-1)`
- `>>> book * 2`
- `>>> book += ['Django','Open']`
- `>>> del book[-1]`

Slicing

- `>>> l = [1,2,3,4,5]`
- `>>> l[0]`
- `>>> l[-1]`
- `>>> l[1:3]`
- `>>> l[:3]`
- `>>> l[1:]`
- `>>> l[:]`

Mapping

- Question
 - How to generate a list with every element multiplied by 2 in a given list ?
- Solution
 - `[2*x for x in list]`

Formatted String & '''

- `>>> s = ' %s is %s ' % ('Django','beautiful')`
- `>>> hi = '''hi`
- `There''' # hi = 'hi\nthere'`
- `>>> print hi`
- `hi`
- `there`

About Tuple

- Unchangeable
- Like list
- Usages

Dictionary

- `>>> book = {'title': 'Python Web Development', 'year': 2008}`
- `>>> 'year' in book`
- `>>> 2008 in book`
- `>>> book['year']`
- `>>> book.get('pub', 'N/A')`
- `>>> book['pub'] = 'Addison Wesley'`
- `>>> del book['pub']`

Built-in & Factory Functions

- Factory Functions (Casting)
 - str
 - list
 - tuple
- Built-in Functions
 - len
 - max
 - range
 - sum

Flow Controls

- Sequence
- Branch
 - if-elif-else
 - Try-raise-except
- Iteration
 - for-in
 - while

Sequence

- The interpreter always interprets codes line by line

Branch

```
# an example of branches
data = raw_input("Enter 'y' or 'n' :")
if data == 'y':
    print 'you typed y'
elif data == 'n':
    print 'you typed n'
else:
    print 'invalid type'
```

Branch

try:

```
f = open('/etc/sudoers')
```

```
...
```

except:

```
print "file doesn't exist"
```

Iteration

```
lst = ['a','b','c','d']
```

```
for i in lst:
```

```
    print i
```

```
dct = {'a':'aa','b':'bb','c':'cc','d':'dd'}
```

```
for k,v in dct:
```

```
    print k,v
```

```
s = 0
```

```
for j in range(101):
```

```
    s += j
```

```
print s
```

Iteration

```
i = 0
```

```
while i < 5:
```

```
    print i
```

```
    i += 1
```

Functions

- Definition

```
def f(n):
```

```
    ...
```

```
    return ...
```

- Call

```
a = f(n)
```

- Warning

- indent

- Value & Reference

Module

- Inside modules
 - functions
 - classes
 - modules
 - variables
- Importing & Loading
 - few restrictions

Module

an example of using modules

- `>>> import copy`
- `>>> a = [1,2,3]`
- `>>> b = copy.copy(a)`
- `>>> b[0]='b'`
- `>>> print a`
- `>>> print b`

OOP

- What is an object?
 - module
 - function
 - list
 - str
 - dog
 - cat
 - ...

Summary

- Variables & Computation
- Flow Controls
- Functions
- Modules
- Little reference of OOP in python