Begining Python[1]

Almark

August 27, 2014

[1] get the latest copy

What is Python

Python is a high-level, interpreted, interactive and object oriengted-scripting language:

- Easy-to-learn
- A broad standard library
- Interactive Mode
- Portable
- Extenable
- Database
- GUI Programming
- Scalable

Python has been widely used in building websites, data mining, machine learning, crawler, etc.

Python Structures

statements
- control flow
- object definitions
- indentation matters - instead of {}

objects
- everything is an object
- automatically reclaimed when no longer needed

modules
- Python source files or C extensions
- import, top-level via from, reload

hello world

```python
# Our first Python program
print('hello world')  # hello world
```

You can write Python code with any editor and execute with command python python-code-file.py or use the Python interpreter in interactive mode

...

basic data types

```
number = 123
string = ’abc’
byte = b’abc’
truth = True
compx = 1 + 2j
nop = None

# id: returns object’s identity.
id(number)
id(compx)

# dir: returns a list of valid attributes
dir(string)
dir(truth)

# type
type(number)  # <class ’int’>
type(compx)  # <class ’complex’>
```

list & tuple

```
# list & tuple
a = [1, 2, 3, 4, 5]
b = list(range(1, 6, 1)) # [1, 2, 3, 4, 5]
c = (1, 2, 3)  # or c = tuple([1, 2, 3])

# list is a natural bidirectional queue
a = [1, 2, 3, 4, 5]
a.pop(0)  # [2, 3, 4, 5]
a.pop()  # [2, 3, 4]
a.insert(0, 7)  # [7, 2, 3, 4]
a.append(8)  # [7, 2, 3, 4, 8]

# slice operation
a = [1, 2, 3, 4, 5]
a[:]  # [1, 2, 3, 4, 5]
a[1:-1:1]  # [2, 3, 4]
a[-1::-2]  # [5, 3, 1]
```

dict

dict in Python is actually a hashtable which is widely used during the runtime. e.g. the variables maintainance.

```
d = {} # or d = dict()
d[1] = 1
d[2] = 4
d[1] # 1
d.get(3, None) # None
d.update({3: 9, 4: 16})

# method items returns the key—value pairs
for k, v in d.items():
  print(k, v)
```

Python has other powerful dicts like OrderedDict, defaultdict provide great features.

set

Python also supports the operations of set. Here is an easy example:

```
p = set('abc')
q = frozenset('cde')


p & q # {'c'}
p | q # {'a', 'e', 'b', 'c', 'd'}
p ^ q # {'e', 'a', 'b', 'd'}
p - q # {'a', 'b'}
```

Control Flow - if Statement

```
a = 1
b = int(input(’enter a number:’))
if b == a:
  print(’equal’)
elif b < a:
  print(’lower’)
else:
  print(’higher’)
```

Unlike C, expressions like a <b <c have the interpretation that is conventional in mathematics:

```
if a < b < c: pass
```

is equivalent to

```
if a < b and b < c: pass
```

Control Flow - loops

```
# for statement
for i in range(0, 5):
  if i == 5:
    print('found the God!!!')
    break
else:
  print('not found')

# while statement
while True:
  pass
else:
  pass
```

The else statement will only be executed when the loop terminates through exhaustion of the list (with for) or when the condition becomes false (with while).

Function

```
def func(a, b=1, *args, **kwargs):
  print(a)
  print(b)
  for arg in args: print(arg)
  for k, v in kwargs.items(): print(k, ’=’, v)
```

different parameters are allowed in Python:

a	The mandatory arguments
b	The arguments with default values
*args	A tuple of the optional arguments
**kwargs	A dict of the optional keyword arguments

how to call:

```
func(’a’, ’b’, 1, 2, 3, c=4, d=5)
```

Anonymous Function

Python supports anonymous function by using the lambda keyword:

```
def make_incrementor(n):
  return lambda x: x + n
f = make_incrementor(42)
f(0)  # 42
f(1)  # 43
```

Another example:

```
processFunc = collapse and (lambda s: " ".join(s.split()
    )) or (lambda s: s)
```

Here the processFunc is determined by the value of collapse.

Exception Handling

Python uses try-except-finally block for exception handling:

```
while 1:
  try:
    x = int(input(’please enter a number:’))
    print(10 / x)
    break
  except ValueError as e:
    print(’Not a valid number’)
    print(e)
  except ZeroDivisionError as e:
    print(’Cannot divide by zero’)
    print(e)
```

You can also process multiple exceptions together:

```
except (ValueError, ZeroDivisionError) as e:
  print(e)
```

Object Oriengted Programming

In Python, everything is an object.

```
class Employee(object):
  def __init__(self, name, empno):
    self.name = name
    self.empno = empno
e = Employee(’Tom’, 101010)
e.empno  # 101010
e.name = ’Tommy’
e.mobile = ’+86 021 12345678’
```

The example above shows how to define an object, it uses __init__ method to initialize the object.

Object Oriengted Programming

Python also supports object inheritance, like Java, we can call super class's __init__. Note here the variable with 2 underlines plays a role of private member.

```
class Manager(Employee):
  def __init__(self, name, empno, band):
    super(Manager, self).__init__(name, empno)
    self.__band = band

  def __str__(self):
    return ’%s, %d’ % (self.name, self.empno)

m = Manager(’Jack’, 123, 9)
m.name   # Jack
m.__band   # ’Manager’ object has no attribute ’__band’
print(m)   # Jack, 123
```

Besides, Python supports multiple inheritance, you can define a class like this:

```
class C(A, B): pass
```

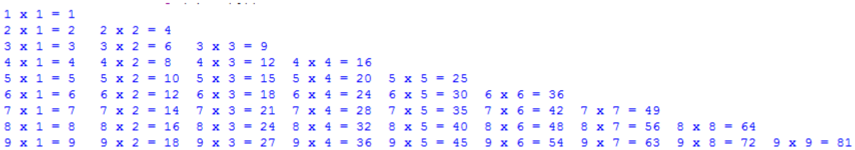
List Comprehensions

List comprehension is a syntactic construct for creating a list based on existing lists.

```
a = [i + 1 for i in range(10)]
a  # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

An example that uses list comprehensions to print a multiplication table

```
print('\n'.join([' '.join(['%d x %d = %-2d' % (i, j, i
    * j) for j in range(1, i + 1)]) for i in range(1, 10)
    ]))
```



```
1 x 1 = 1
2 x 1 = 2   2 x 2 = 4
3 x 1 = 3   3 x 2 = 6   3 x 3 = 9
4 x 1 = 4   4 x 2 = 8   4 x 3 = 12  4 x 4 = 16
5 x 1 = 5   5 x 2 = 10  5 x 3 = 15  5 x 4 = 20  5 x 5 = 25
6 x 1 = 6   6 x 2 = 12  6 x 3 = 18  6 x 4 = 24  6 x 5 = 30  6 x 6 = 36
7 x 1 = 7   7 x 2 = 14  7 x 3 = 21  7 x 4 = 28  7 x 5 = 35  7 x 6 = 42  7 x 7 = 49
8 x 1 = 8   8 x 2 = 16  8 x 3 = 24  8 x 4 = 32  8 x 5 = 40  8 x 6 = 48  8 x 7 = 56  8 x 8 = 64
9 x 1 = 9   9 x 2 = 18  9 x 3 = 27  9 x 4 = 36  9 x 5 = 45  9 x 6 = 54  9 x 7 = 63  9 x 8 = 72  9 x 9 = 81
```

Decorator

You may have been familiar with the decorator pattern, Python provides a more simple but powerful decorator in language level. An example of log function

```
def log(fn):
  def wrapper():
    print(’start executing, %s’ % fn.__name__)
    fn()
    print(’end executing, %s’ % fn.__name__)
  return wrapper

@log
def foo():
  print(’I am foo’)

foo()
```

Decorator - An Aadvanced Example

```python
from functools import wraps
def memo(fn):
  cache = {}
  miss = object()
  @wraps(fn)
  def wrapper(*args):
    result = cache.get(args, miss)
    if result is miss:
      result = fn(*args)
      cache[args] = result
    return result
  return wrapper

@memo
def fib(n):
  if n < 2: return n
  return fib(n - 1) + fib(n - 2)
```

Generator

Generators functions allow you to declare a function that behaves like an iterator, i.e. it can be used in a for loop.

```python
def get_next_prime():
  yield 2
  yield 3

  ret = 4
  while True:
    ret += 1   # starts from 5
    for i in range(2, ret - 1):
      if ret % i == 0: break
    else: yield ret

prime_generator = get_next_prime()
next(prime_generator)   # 2
next(prime_generator)   # 3
for v in get_next_prime():
  print(v)   # endless
```

Concurrent Programming

4 types of concurrent programming in Python:

- multi-processing os.fork, multiprocessing
- multi-threading threading, Thread
- asynchronous select, poll, epoll (depends on OS)
    - coroutine yield, asyncio (Python 3.4)

Co-Operative Routines

Coroutines are program components that generalize subroutines to allow multiple entry points for suspending and resuming execution at certain locations.

```
import sys
def produce(l, top):
  i = 0
  while i < top:
    l.append(i)
    yield i
    i = i + 1

def consume(l, top):
  p = produce(l, 10)
  while 1:
    try:
      next(p)
      while len(l) > 0: print(l.pop())
    except StopIteration: sys.exit(0)

consume([], 10)
```

Some powerful 3rd party modules

gevent

gevent is a coroutine-based Python networking library that uses greenlet to provide a high-level synchronous API on top of the libev event loop. Below is a simple example shows the producer-consumer model

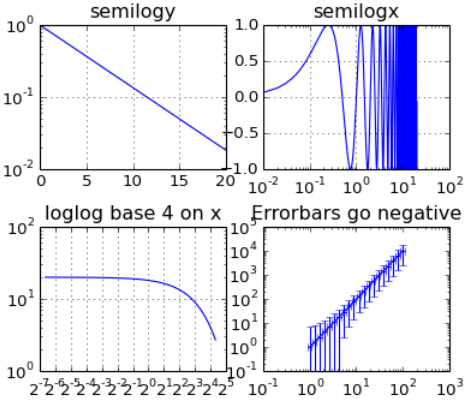
```
import gevent
from gevent.queue import Queue

tasks = Queue()
def worker(n):
    while not tasks.empty():
        task = tasks.get()
        print(’Worker %s got task %s’ % (n, task))
        gevent.sleep(0)
def boss():
    for i in xrange(1,25): tasks.put_nowait(i)

gevent.spawn(boss).join()
gevent.joinall([
    gevent.spawn(worker, ’steve’),
    gevent.spawn(worker, ’john’),
])
```

matplotlib

matplotlib is python 2D plotting library with a set of API which is similar to matlab. Below is a demo from official website.



Pony ORM

Pony is a cool and new Python ORM that lets you query a database using Python generators. These generators are then translated into effective SQL.
Python generator:

```
select(c for c in Customer if sum(c.orders.price) >
    1000)
```

is translated to following SQL:

```
SELECT "c"."id"
FROM "Customer" "c"
  LEFT JOIN "Order" "order-1"
    ON "c"."id" = "order-1"."customer"
GROUP BY "c"."id"
HAVING coalesce(SUM("order-1"."total_price"), 0) > 1000
```

References

- Pro Python, a book introduces advanced usage of Python
- TimeComplexity: https://wiki.python.org/moin/TimeComplexity
- Python 2 or Python 3: https://wiki.python.org/moin/Python2orPython3
- Method Resolution Order (MRO): https://www.python.org/download/releases/2.3/mro/
- List Comprehensions: http://legacy.python.org/dev/peps/pep-0202/
- Tasks and coroutines: https://docs.python.org/3/library/asyncio-task.html