第六讲 Python 语言介绍

清华大学计算机系

课程提纲

- 为什么介绍Python
- 什么是Python
- Python优点
- Python 2.7.15
- Python 3.5.6
- 课程大作业和考试不限Python 2和3
- Python 2和3的区别
  - ✓ https://chenqx.github.io/2014/11/10/Key-differences-between-Python-2-7-x-and-Python-3-x/

为什么介绍Python

- 掌握接受新语言的能力
- 熟悉其他类型语言
- 增强解决问题的能力

什么是Python

- 创始人 **Guido van Rossum**
- 始于**1989**
- 大蟒蛇
- **Python**优点
  - ✓简单易学，功能强大的编程语言
  - ✓高效率的高层数据结构
  - ✓简单而有效地实现面向对象编程
  - ✓语法简洁，支持动态输入
  - ✓跨平台的理想的脚本语言
  - ✓适用于快速的应用程序开发

程序设计语言占有率

Jun 2018	Jun 2017	Change	Programming Language	Ratings	Change
1	1		Java	15.368%	+0.88%
2	2		C	14.936%	+8.09%
3	3		C++	8.337%	+2.61%
4	4		Python	5.761%	+1.43%
5	5		C#	4.314%	+0.78%
6	6		Visual Basic .NET	3.762%	+0.65%
7	8	^	PHP	2.881%	+0.11%
8	7	v	JavaScript	2.495%	-0.53%
9	-	^^	SQL	2.339%	+2.34%
10	14	^^	R	1.452%	-0.70%
11	11		Ruby	1.253%	-0.97%
12	18	^^	Objective-C	1.181%	-0.78%
13	16	^	Visual Basic	1.154%	-0.86%
14	9	vv	Perl	1.147%	-1.16%
15	12	v	Swift	1.145%	-1.06%
16	10	vv	Assembly language	0.915%	-1.34%
17	17		MATLAB	0.894%	-1.10%
18	15	v	Go	0.879%	-1.17%
19	13	vv	Delphi/Object Pascal	0.875%	-1.28%

为什么用Python

- 简单易学
- 代码优美
- 轻量级开发工具
- 类库丰富

什么人用Python

- **NASA(美国国家航空航天局)既用Python做系统开发，又将其作为脚本语言**
- **Industrial Light & Magic在高预算影片中使用Python制作影片的特效**
- **Yahoo!使用Python（包括其他技术）管理讨论组**
- **Google用Python实现Web爬虫和搜索引擎中的很多组件**

Python适用的场合

- 数值计算
  - ✓2^{100}
- 网页处理
  - ✓网页爬虫
- 文本处理
  - ✓**HTML parser**
- 脚本程序
  - ✓宏或者批处理
- 新手入门

适合学习对象

- 软件开发人员
- 网站运维人员
- 高级动画设计人员
- 系统原型架构设计人员

Python 特点

- 面向对象
- 平台无关
- 强类型语言
- 动态语言
- 解释性语言
- 脚本语言

弱类型 vs 强类型

- 强类型
  - ✓强制数据类型定义的语言，除了强制转换，数据类型不能改变
  - ✓**int char float**
  - ✓例如 **Python, C, C++, Java, …**
- 弱类型
  - ✓数据类型可以被忽略的语言
  - ✓**a=“1”， b=2**
  - ✓**a+b**
  - ✓例如 vbscript，**php…**

动态语言 vs 静态语言

- 动态类型语言
  - ✓在运行期间才去做数据类型检查的语言
  - ✓在用动态类型的语言编程时，永远也不用给任何变量指定数据类型，该语言会在你第一次赋值给变量时，在内部将数据类型记录下来
  - ✓例如 Python
- 静态类型语言
  - ✓数据类型是在编译其间检查的，也就是说在写程序时要声明所有变量的数据类型
  - ✓例如C、C++、Java

解释性 vs 编译性语言

- 高级语言翻译成机器语言
- 翻译模式
  - ✓解释性
    - 边翻译边执行，运行速度慢
  - ✓编译性
    - 首先翻译，优化代码，运行较快

脚本语言

- 控制软件应用程序
- 通常以文本(如ASCII)保存，只在被调用时进行解释或编译
- 优点
  - ✓良好的快速开发
  - ✓高效率的执行
  - ✓解释而非编译执行
  - ✓和其它语言编写的程序组件之间通信功能很强大。

Python优势

- 可读性好
- 可嵌入性 （胶水语言）
- 简单易学
- 网络编程
- 游戏编程

Python劣势

- 性能差
- 缺乏真正的多处理器支持
- 缺乏商业支持

Python运行环境

Python 运行环境

1 下载Python编译器

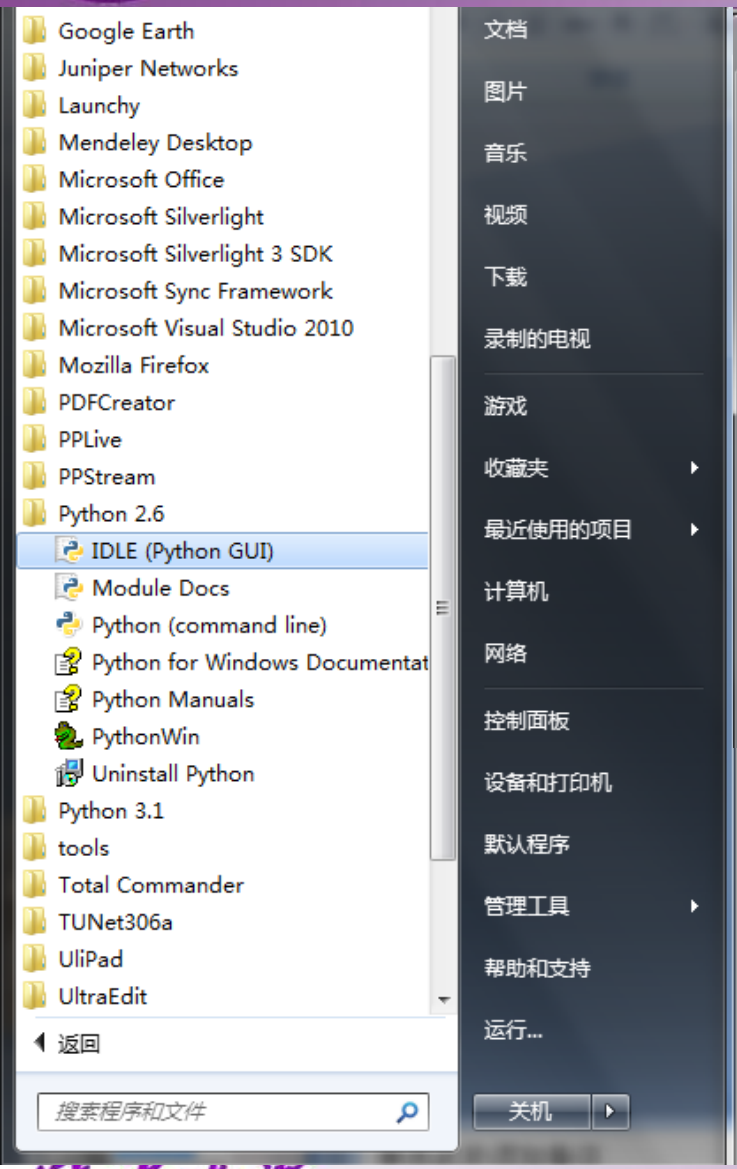
https://www.python.org/downloads/

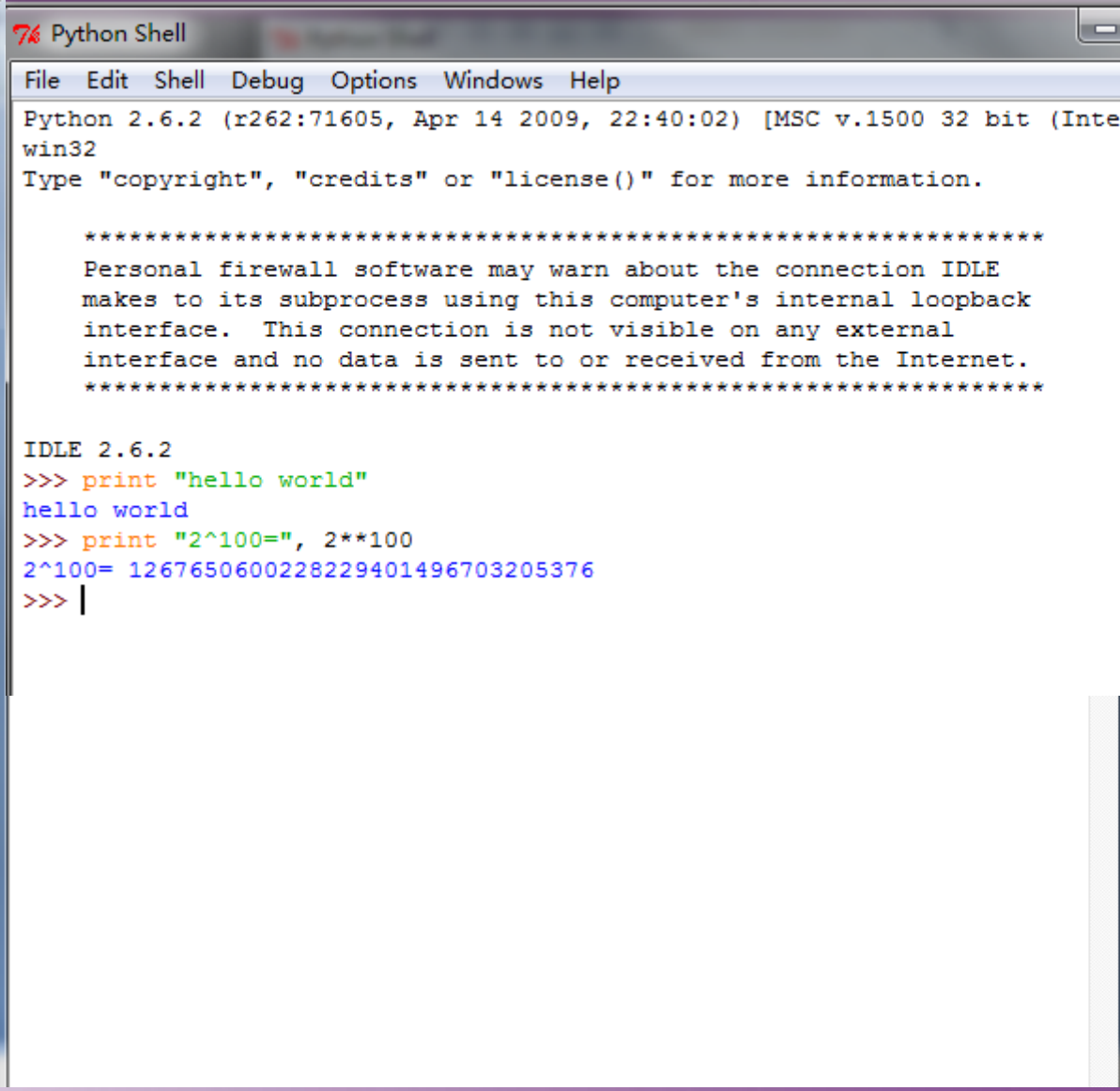
2 安装

3 编写“Hello World！”程序

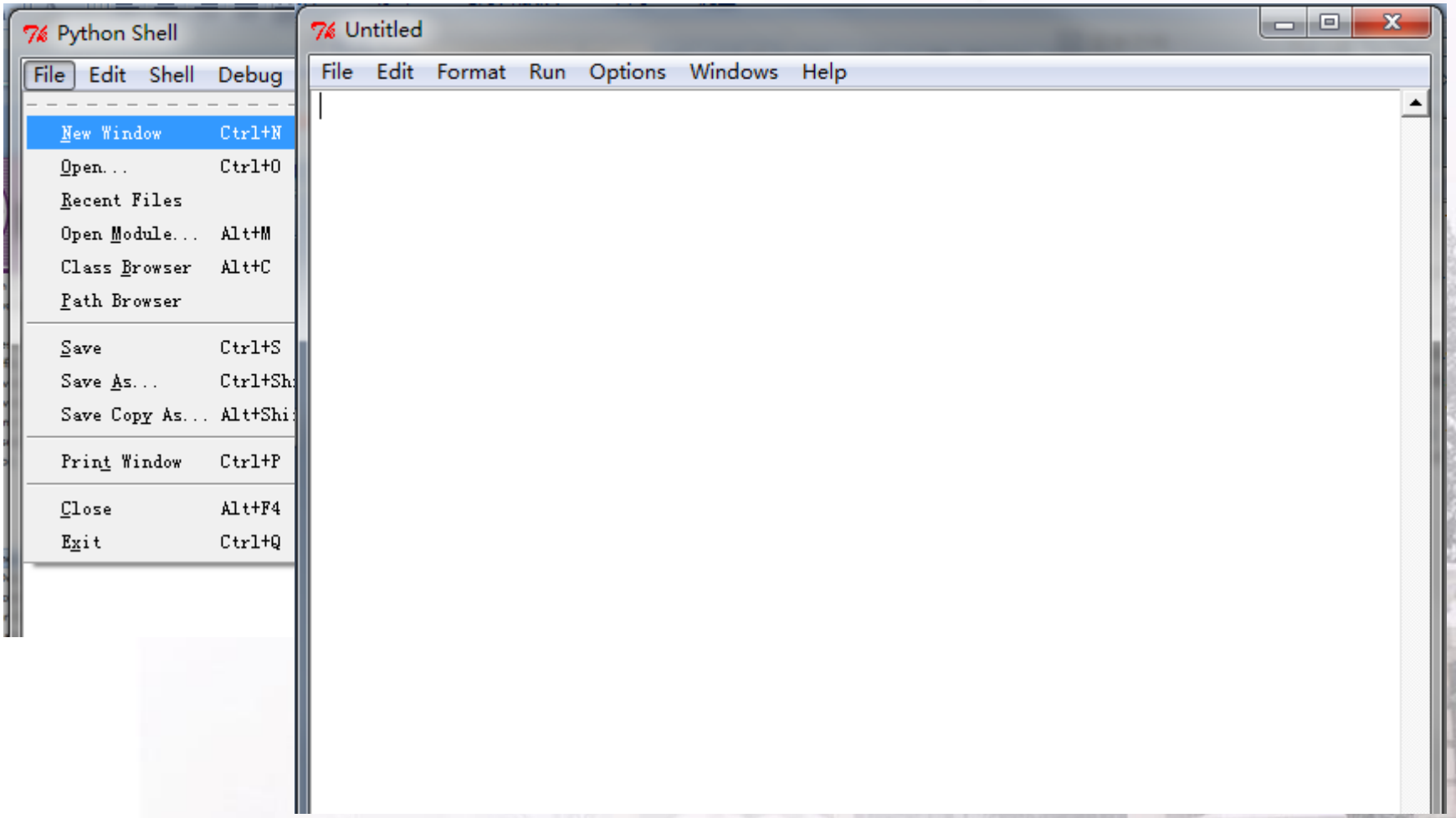
- ✓创建文件 hw.py
  - □输入 print "Hello World!"
- ✓保存
- ✓命令行执行python hw.py

Python 工具 IDLE





创建Hello World程序



编写Hello World程序

File Edit Format Run Options Windows Help

```
print "Hello World!"
```

File Edit Format Run Options Windows Help

```
print "Hello Wo
```

Python Shell

Check Module Alt+X

Run Module F5

```
print "Hello World!"
```

Python Shell

File Edit Debug Options Windows Help

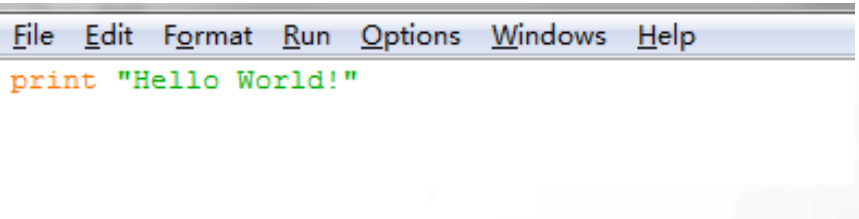
```
Python 2.6.2 (r262:71605, Apr 14 2009, 22:40:02) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.

    **********************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    **********************************************************************

IDLE 2.6.2      ==== No Subprocess ====
>>>
Hello World!
>>>
```

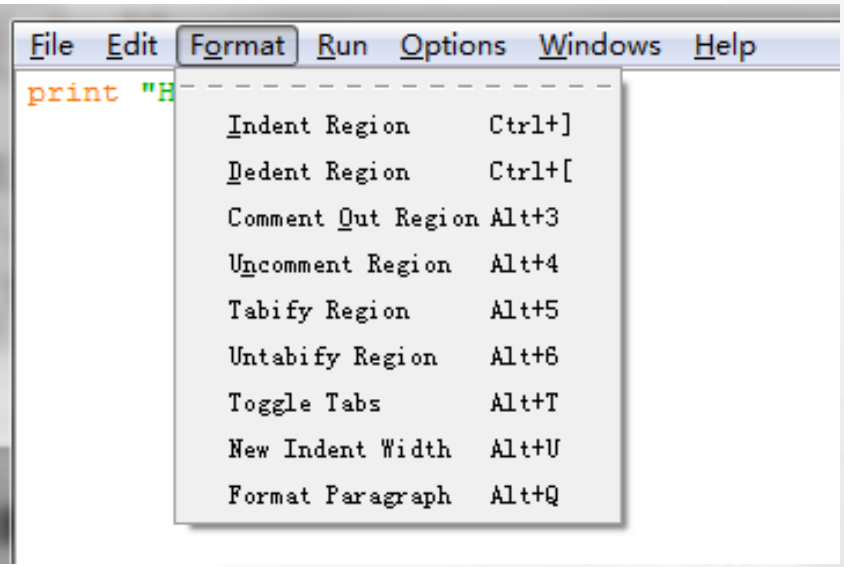
Python IDE工具介绍

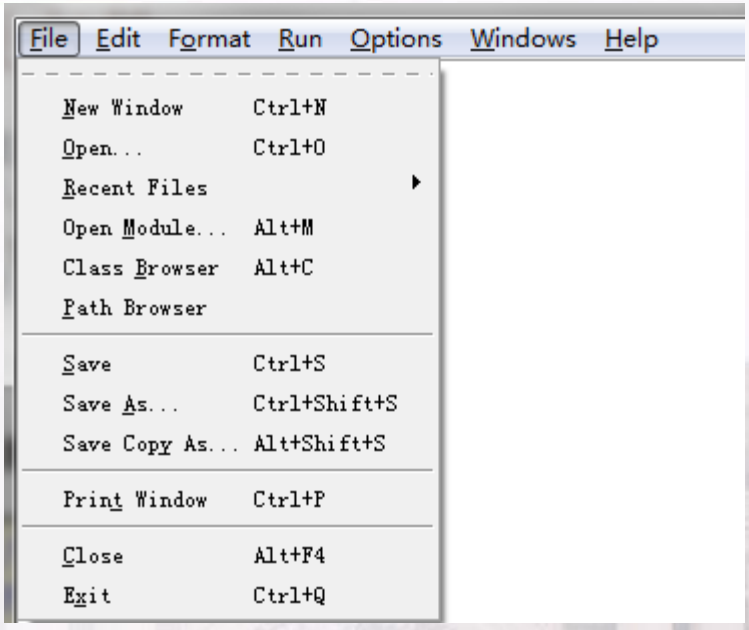
- Windows
  - ✓IDLE
    - Python自带
  - ✓Ulipad
    - http://code.google.com/p/ulipad/downloads/list
- Linux
  - ✓Vim
  - ✓Emacs

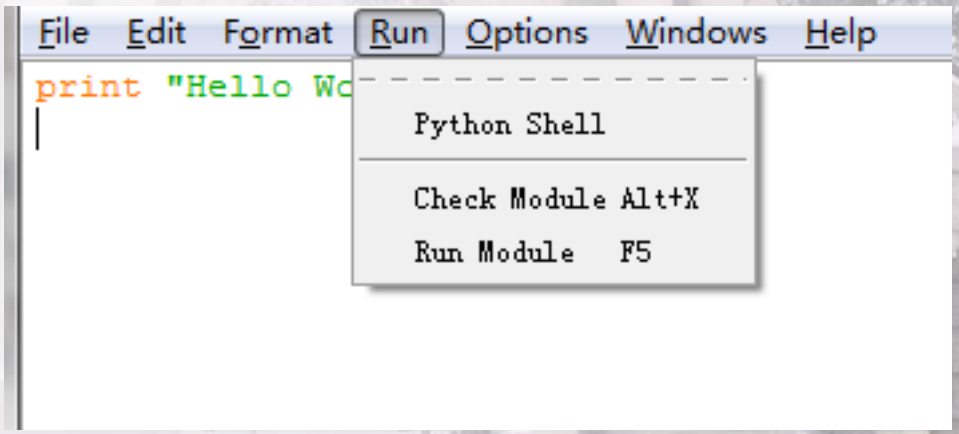
IDLE

Python 自带

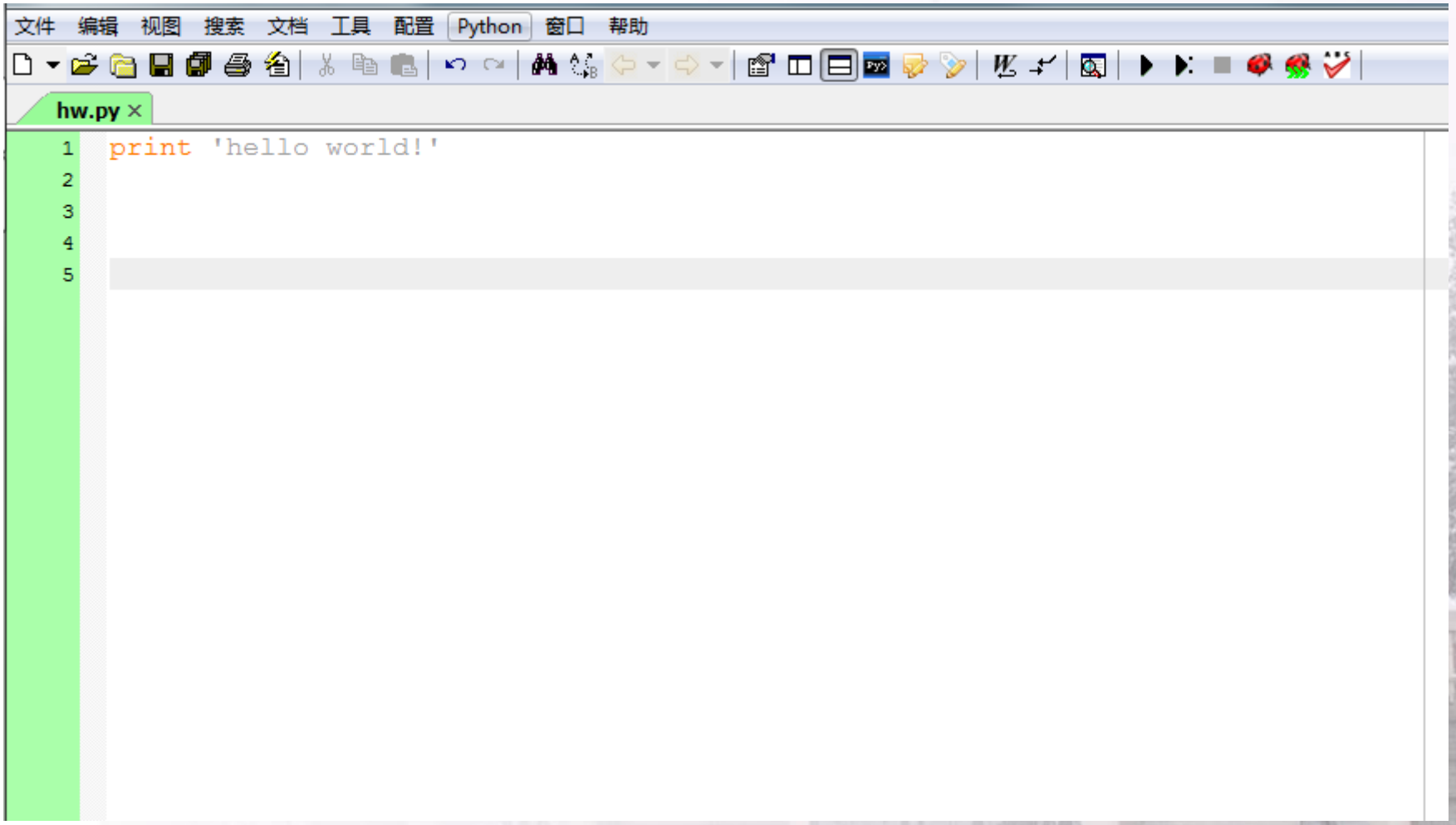








UliPad



Python 数据类型

数值类型

- 整数
  - ✓2， 3
- 长整数
  - ✓1234567890123
- 浮点数
  - ✓3.23， 3.23E-4
- 复数
  - ✓4-5j

```
IDLE 2.6.2
>>> a = 2
>>> print a
2
>>> type(a)
<type 'int'>
>>> a = 2.0
>>> print a
2.0
>>> type(a)
<type 'float'>
>>> a = 3.14E5
>>> print a
314000.0
>>> type(a)
<type 'float'>
>>> a = 2.43E-5
>>> print(a)
2.43e-05
>>> type(a)
<type 'float'>
>>> a = 4+5j
>>> print a
(4+5j)
>>> type(a)
<type 'complex'>
>>>
```

运算符

- 运算符
  - ✓+ - * /
  - ✓** （幂运算符）
  - ✓% （模除）
  - ✓< > = != == >= <=
  - ✓<< >>
  - ✓& |
- 运算符优先级

or	布尔“或”
and	布尔“与”
not x	布尔“非”
in, not in	成员测试
is, is not	同一性测试
<, <=, >, >=, !=, ==	比较
|	按位或
^	按位异或
&	按位与
<<, >>	移位
+, -	加法与减法
*, /, %	乘法、除法与取余
+x, -x	正负号
~x	按位翻转
**	指数
x.attribute	属性参考

布尔型

- True 1
- False 0
- 0 -> False 其他数都是 True
- 布尔操作 and or not
- 例子
  - ✓True*12 = 12
  - ✓0 and 1 = 0
  - ✓0 or 2 = 2
  - ✓12 < 13 True

不同数据类型的布尔值

int	**0**	**False**
	-1	True
	124	True
float	**0.0**	**False**
str	**""**	**False**
	"False"	True
dict	**{ }**	**False**
	{'key': 'val'}	True
list	**[ ]**	**False**
	[False]	True

字符串

- 字符串
  - ✓‘hello’
  - ✓“hello”
  - ✓“‘ hello,
    world”’
  - ✓字符串不能更改
  - ✓转义
    - \’
  - ✓使用原始串，不转义
    - r‘hello\nworld’

可换行
程序注释

```
IDLE 2.6.2
>>> print "hello world"
hello world
>>> a = "hello world!"
>>> print a
hello world!
>>> type(a)
<type 'str'>
>>> a = ''' hello
world!
'''
>>> print a
 hello
world!

>>> type(a)
<type 'str'>
>>>
```

类型转换

函数	描述
int(x[,base])	将x转换为一个整数
long(x[,base])	将x转换为一个长整数
float(x)	将x转换到一个浮点数
complex(real [,imag])	创建一个复数
str(x)	将对象 x 转换为字符串
repr(x)	将对象 x 转换为字符串
chr(x)	将一个整数转换为一个字符
unichr(x)	将一个整数转换为Unicode字符
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

str() 和repr() 区别

- str() 人可以读懂的字符串
  - ✓一般用于整数和浮点数
- repr() 解释器识别的字符串
  - ✓一般用于对象

类型转换的例子

```
>>> a = "58"
>>> type(a)
<type 'str'>
>>> b=int(a)
>>> print b
58
>>> type(b)
<type 'int'>
```

```
>>> f = float('1.2e-3')
>>> print f
0.0012
>>> eval('23-12')
11
```

eval 用来计算Python表达式的结果

动态性

- exec("a=2")
- exec("b=1")
- print eval("a+b")
  - ✓3
- execfile('hw.py')
  - ✓执行hw.py 文件

字符串操作

```
>>> a = "Part 1"
>>> b = "and part 2"
>>> a + ' ' + b
'Part 1 and part 2'
>>> s = a * 2
>>> print s
Part 1Part 1
```

```
>>> s[0]
'P'
>>> s[0:4]
'Part '
>>> s[5:]
'1Part 1'
>>> s[6:-1]
'Part '
```

字符串不能修改

- s=‘hello world’
- s[0]=‘H’ 错误

- s= ‘H’+s[1:] 正确

字符串函数 (1)

- count　出现次数
  - ✓s = 'a string, with stuff'
  - ✓s.count('st')
  - ✓2
- len　字符串长度
  - ✓s = 'a string, with stuff'
  - ✓len(s)
  - ✓20

字符串函数 (2)

- rjust 右对齐填充
  - ✓s = 'a string, with stuff‘
  - ✓s.rjust(30)
  - ✓s='　　a string, with stuff‘
- ljust 左对齐填充
  - ✓s = 'a string, with stuff‘
  - ✓s.ljust(30)
  - ✓s='a string, with stuff　　‘
- center 填充两侧

字符串函数 (3)

- upper、lower 大小写
  - ✓ s=“Super”
  - ✓ s.upper() s.lower()
- split 切分
  - ✓ s="I am python; do you love me?"
  - ✓ s.split() → ['I', 'am', 'python;', 'do', 'you', 'love', 'me?']
- join 连接
  - ✓ s1=":"
  - ✓ s2=('0','1','2','3‘)
  - ✓ print s1.join(s2) → 0:1:2:3

字符串函数 (4)

- find, rfind, index, rindex 查找
  - ✓s="I am python; do you love me?"
  - ✓s.find("do") → 13
  - ✓找不到find 返回-1；index返回错误
- replace 替换
  - ✓s="I am python; do you love me?"
  - ✓s.replace("me","python") → 'I am python; do you love python?'
- isdigit(), islower(), isupper() 大小写等
- startswith, endswith 开始于, 结束于

字符串函数 (5)

- strip(), lstrip(), rstrip()
  - ✓除去空白字符 包括 \r\t\v\f\n
  - ✓s=" I am python; do you love me? \n"
  - ✓s.strip() →'I am python; do you love me?'
- title() 首字母大写
  - ✓s=" I am python; do you love me? "
  - ✓s.title()→' I Am Python; Do You Love Me? '
- encode, decode 转换为unicode

字符串格式化

格式化方法

- %s: string (uses function 'str')
- %r: string (uses function 'repr')
- %i: int
- %f, %e, %g: float

➢ `w = "Number %i won!" % 12`

✓ `Number 12 won!`

➢ `w = "Number %f won!" % 11.5`

✓ `Number 11.5 won!`

➢ `w = "Number %i %f won!" % (12, 11.5)`

✓ `Number 12 11.5 won!`

字符串格式化码

格式	描述
%%	百分号标记
%c	字符及其ASCII码
%s	字符串
%d	有符号整数(十进制)
%u	无符号整数(十进制)
%o	无符号整数(八进制)
%x	无符号整数(十六进制)
%X	无符号整数(十六进制大写字符)
%e	浮点数字(科学计数法)
%E	浮点数字(科学计数法，用E代替e)
%f	浮点数字(用小数点符号)
%g	浮点数字(根据值的大小采用%e或%f)
%G	浮点数字(类似于%g)
%p	指针(用十六进制打印值的内存地址)

列表 List

- **shoplist = ['apple', 'mango', 'carrot', 'banana']**
- 列表常用方法
  - ✓ 定位 **list[n]**
    - **shoplist[1] → 'mango'**
  - ✓ 复制 **shoplist*2 → ['apple', 'mango', 'carrot', 'banana', 'apple', 'mango', 'carrot', 'banana']**
  - ✓ 加法 **shoplist+['papaya']**

  → ['apple', 'mango', 'carrot', 'banana', 'papaya']
  - ✓ 遍历
    - **for item in shoplist:**
      print item

列表List

- 负数索引
- 0 是第一个元素
- -1 是最后一个元素
- -i 和 len-i是同一个元素

列表方法

- **append(value), extend**
  - ✓ **r.append(['1', '2'])**
    - [r, ['1', '2']]
  - ✓ **r.extend(['1', '2'])**
    - [r, '1', '2']
  - ✓ **r.extend 等同于 运算符+**
    - r+['1', '2']]=r.extend(['1', '2'])
- **insert(index, value)**
  - ✓ **r.insert(0,3), r.insert(0,'3')**
    - → [3, r] → ['3', r]

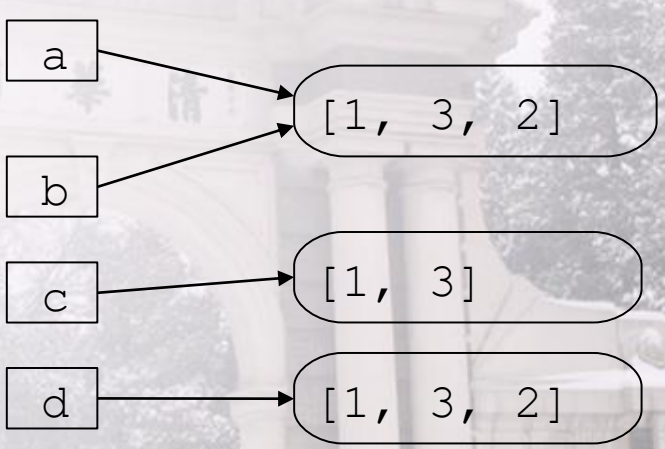
列表方法

- 删除
  - ✓ remove (value)
  - ✓ r.remove(1) 从r中删除1
- 搜索
  - ✓index(value)
  - ✓r.index(1) 查找1的位置， 多个值返回第一个
- 倒转
  - ✓reverse()
- 排序
  - ✓sort ()

拷贝

- 对于列表、字典等类型（不包含数值类型和字符串类型）
  - ✓赋值代表引用，而不是拷贝

```
>>> a = [1, 3, 2]
>>> b = a
>>> c = b[0:2]
>>> d = b[:]
```




```
>>> b.sort()
>>> a
[1, 2, 3]
```

相等性比较

```
>>> a = [1, 2]
>>> b = [1, 2]
>>> a == b          # test whether values are equal
True
>>> a is b          # test whether objects are identical
False
>>> a.append(3)
>>> a == b          # test values again
False
```

元组 Tuple

- 和list 类似，但是不能修改
- zoo = ('wolf', 'elephant', 'penguin')
- 可以嵌套定义：

  new_zoo = ('monkey', 'dolphin', zoo)
- 元组没有方法
- List转换为tuple：
  - ✓l=[1,2,3]
  - ✓t=tuple(l) → (1,2,3)
  - ✓l=list(t) → [1,2,3]

Tuple例子

- `t = (1, 3, 2)`
- `(a, b, c) = t`
- `a, b = b, a` → `a=3 b=1`
- `r = list(t)` → `[1,3,2]`
- `tuple(r)` → `(1, 3, 2)`

字典 Dictionary

- Key/value对
- d = { 'Swaroop' : 'swaroopch@byteofpython.info',
  'Larry' : 'larry@wall.org',
  'Matsumoto' :'matz@ruby-lang.org',
  'Spammer':'spammer@hotmail.com'}

- 查找key值 d['Larry']= 'larry@wall.org'
- 查找key d.has_key('Larry') = True
- 添加 d['lucy']='lucy@gmail.com'
- 删除 del d['Larry']

字典常用方法

- 清空 clear()
- 复制 copy(), deepcopy()
  - ✓ d = {'age' : 12, 'name' : 'bob'}
    y = d.copy()
    y['age'] = 32
    d={'age': 12, 'name': 'bob'}
    y={'age': 32, 'name': 'bob'}

```
d = {'names': ['bob', 'sam']}
y = d.copy()
y['names'][0] = 'jack'
y={'names': ['jack', 'sam']}
d={'names': ['jack', 'sam']}
```

```
import copy
d = {'names': ['bob', 'sam']}
y = copy.deepcopy(d)
y['names'][0] = 'jack'
d={'names': [‘bob', 'sam']}
y={'names': ['jack', 'sam']}
```

字典常用方法（2）

- 长度 len()
- 获取键、值、键值对
  - ✓Keys()
  - ✓Values()
  - ✓Items()

输入输出

```
s=raw_input("pls input an integer:")

i= int(s)

print i
```

Python 变量和语句

变量

- ✓标识符的第一个字符
  - 必须是字母表中的字母（大写或小写）
  - 或者一个下划线（‘_’）。
- ✓标识符名称的其他部分
  - 可以由字母（大写或小写）
  - 下划线（‘_’）
  - 数字（0-9）组成。
- ✓标识符名称是对大小写敏感的。
  - 例如，**myname**和**myName**不是一个标识符。
  - 有效标识符：**I _my_name name_23 a1b2_c3**。
  - 无效标识符：**2things**、**this is spaced out my-name**。

Python变量不指定类型

- ✓a=1
- ✓b=5.2
- ✓c=“abc”
- ✓list=['a','b','c','d','e','f','g']
- ✓tuple=(1,2,3)
- ✓d=word[1:3]
- ✓e=word[:2]
- ✓f=word[0:]
- ✓g=word[-4:-2]

代码要求

- 逻辑行与物理行
  - ✓i = 5 逻辑行
    print i
  - ✓i = 5； print i ； 物理行

- 缩进
  - ✓同一层次的语句必须相同的缩进
    - i=5
    - j=6 错误
  - ✓建议一致TAB或者SPACE

控制流

- If 语句
  - ✓ if a==b:
  - ✓ elif a>=b:
  - ✓ else:
- While语句
  - ✓ while i<2:
- For语句
  - ✓ for i in [1,2,3,4,5]:
  - ✓ for c in ‘hello python':
- Break语句
  - ✓ while true:
    - ✓ If s==2:
      - ✓ break
- Continue语句
  - ✓ while true:
    - ✓ If s==2:
      - ✓ continue

断行

- 使用\

```
if a_complicated_expression and \
  another_complicated_expression:
    print 'this is valid syntax'
```

- 使用()

```
if (a_complicated_expression and
  another_complicated_expression):
    print 'this is valid syntax'
```

Python函数

函数

- 函数定义

```
def sum(a,b) :
    return a+b
```

- 函数调用

```
func = sum （函数也可以赋值）
r = func(5,6)
```

带参数的函数

- 参数个数固定

```
def add(a,b):
    return a+b

r=add(1,5)
print r  → 6
```

默认参数

- def say(message, times = 1):
    print message * times

say('World', 5) → 'WorldWorldWorldWorldWorld'

关键参数

- 根据参数的名字进行参数传递

- **def func(a, b=5, c=10):**
  print 'a is', a, 'and b is', b, 'and c is', c

  func(3, 7) → a=3, b=7, c=10
  func(25, c=24) → a=25, b=5, c=24
  func(c=50, a=100) → a=100, b=5, c=50

元组参数

*args可以代表多个参数

```
def noargs(a, *args):
    print "a=%s, others=%s" % (a, args)

noargs("hello", 1, 2, 3, "python", "good")
```

→ a=hello, others=(1, 2, 3, 'python', 'good')

字典参数

```
def keyword_args(a, b='bla', **kwargs):
    return "a=%s, b=%s, kwargs=%s" % (a, b, str(kwargs))

keyword_args(c='call', d=12, a='gr')
```

```
a=gr, b=bla, kwargs={'c': 'call', 'd': 12}
```

参数规则

- 默认参数必须在非默认参数之后
- 只能用一个元组参数和一个字典参数
- 元组参数必须在默认参数之后（*arg）
- 字典参数必须在最后（**arg）

函数示例

```
def testfun(fixed1,fixed2,key1=1,key2=2,*arg,**keywords):
    print "fixed1 parameters is ",fixed1
    print "fixed2 parameters is ",fixed2
    print "key1 parameter is ",key1
    print "key2 parameter is ",key2
    print "Arbitrary parameter is ", arg
    print "keywords parameter is ",keywords
testfun(1,2,3,4,5,6,k1=1,k2=2,k3=3)
```

```
fixed1 parameters is  1
fixed2 parameters is  2
key1 parameter is  3
key2 parameter is  4
Arbitrary parameter is  (5, 6)
keywords parameter is  {'k3': 3, 'k2': 2, 'k1': 1}
```

Lambda 函数

- Lambda 定义单行最小函数
- 作用类似于宏定义

g = lambda x: x*2

g(3) = 6

(lambda x,y:x+y)(2,3) → 5

局部变量

- 局部变量作用域 – 函数体内部

```
def test_local(a, r):
    print 'local original ', a, r
    a = 12
    r[1] = 999
    print 'local changed  ', a, r

a = -5
r = [0, 1, 2]
print 'global original', a, r
test_local(a, r)
print 'global changed ', a, r
```

r是对象

```
global original -5 [0, 1, 2]
local original  -5 [0, 1, 2]
local changed   12 [0, 999, 2]
global changed  -5 [0, 999, 2]
```

全局变量

```
def func():
    global x
    print 'x is', x
    x = 2
    print 'Changed local x to', x

x = 50
func()
print 'Value of x is', x
```

Return语句

```
def maximum(x, y):
    if x > y:
        return x
    else:
        return y

print maximum(2, 3)
```

包机制

- a.py
  - ✓def sum(a,b):
  - ✓ return a+b;

- b.py
  - ✓from a import sum
  - ✓Print "3+2=",sum(3,2)

DocStrings

- 使得程序更加易懂
- def printMax(x, y):

```
    '''Prints the maximum of two numbers.
    The two values must be integers.'''
    x = int(x)
    y = int(y)
    if x > y:
        print x, 'is maximum'
    else:
        print y, 'is maximum'
print printMax.__doc__
printMax(3, 5)
```

创建自己的包

```
def sayhi():
    print 'Hi, this is mymodule speaking. ’
version = '0.1'

from mymodule import sayhi, version
    sayhi()
    print 'Version', version
```

Python面向对象编程

面向对象的优点

- 可扩展性：
  - ✓可扩展性强
- 重用性：
  - ✓代码重用性强
- 灵活性：
  - ✓维护简单
  - ✓结构清晰、标准、规范化
  - ✓易于理解
  - ✓可读性更强

面向对象的特点

- 抽象
- 封装
- 继承
- 多态

类和对象

- 类
  - ✓对象的抽象
  - ✓包括属性和方法
- 对象
  - ✓类具体的实例

类例子

geom1.py

```
import math
class Circle:
    '''A 2D circle.'''
    def __init__(self, x, y, radius=1):
        self.x = x
        self.y = y
        self.radius = radius
    def area(self):
        ''' Return the area of the shape. '''
        return math.pi * self.radius**2
```

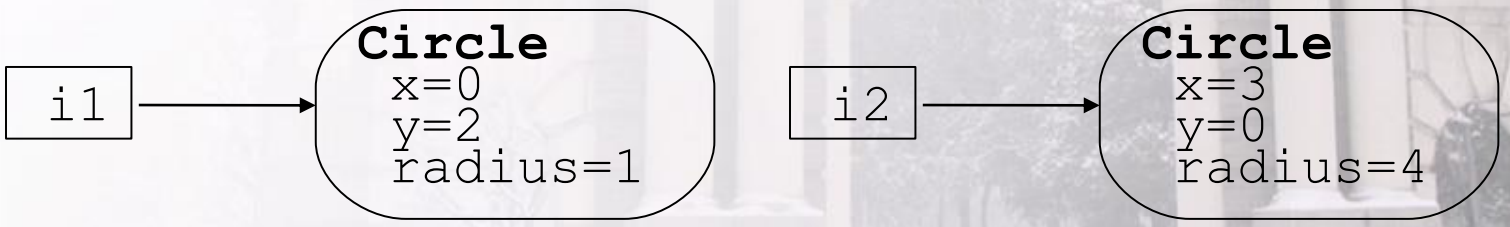
构造函数
this 指针

Python类

- 成员变量
  - ✓self: this 指针
- 成员函数
  - ✓__init__：构造函数
  - ✓__del__：析构函数

Python对象

```
from geom1 import *
i1 = Circle(0, 2)
i2 = Circle(3, 0, 4)
print 'i1:', i1.radius, i1.area()
print 'i2:', i2.radius, i2.area()
```



属性值可以改变

```
from geom1 import *

i1 = Circle(0, 2)
print 'i1:', i1.radius, i1.area()

i1.radius = 2.5
print 'i1:', i1.radius, i1.area()
```

属性值可以增加删除

```
from geom1 import *

i1 = Circle(0, 2)
i1.color =  'red'

del i1.radius
```

类属性和方法

- 私有属性 __attr 加双下划线
  - ✓不能删除、修改
- 普通类方法只有对象（类实例）可见
- 静态方法 staticmethod对类和对象可见
  - ✓@staticmethod
  - ✓def sm
  - ✓不接受一个隐式的第一个参数
- 类方法 classmethod 对类和对象可见
  - ✓@classmethod
  - ✓def cm(cls)
  - ✓接受一个隐式的第一个参数cls

例子

```
class p:
    z=0
    def __init__(self,x,y,z=50):
        self.x=x
        self.y=y
        self.__z=5
        p.z=z
    def pout(self):
        print 'self.x=', self.x
        print 'self.y=', self.y
        print 'self.__z=', self.__z
        print 'p.z=', p.z
        print "\n”
```

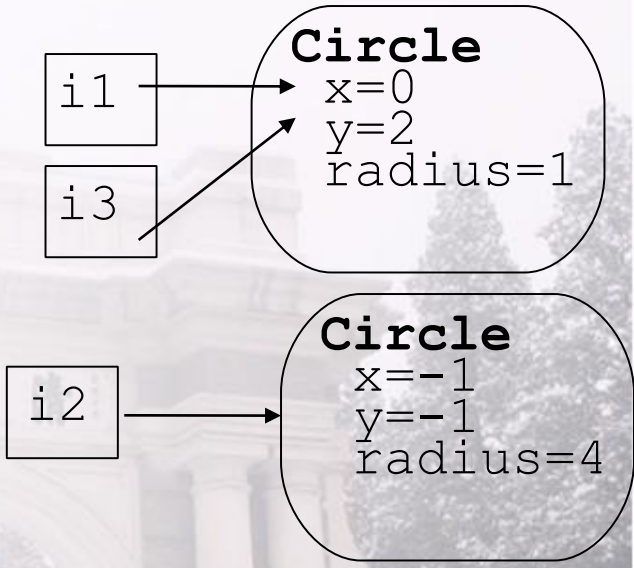
```
@staticmethod
def pstatic():
    print "i am staticmethod"
    p.z=p.z+1
@classmethod
def pclass(cls):
    print "i am classmethod"
    p.z=p.z+2
```

```
a = p(1,2)      p.pclass()
a.pout()        a.pout()
a.x=3           p.pstatic()
a.pout()        a.pout()
```

p.pout() error

引用与拷贝

```
from geom1 import *
i1 = Circle(0, 2)
i2 = Circle(-1, -1, 4)
i3 = i1
i1.radius = 1.75
print 'i1:', i1.radius
print 'i2:', i2.radius
print 'i3:', i3.radius
```



继承

```
import math
class Shape:
    def is_round(self):
        return True

class Circle(Shape):

class Blob(Shape):
    def is_round(self):
        return false
```

类例子

```
class SchoolMember:
    '''Represents any school member.'''
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print '(Initialized SchoolMember: %s)'% self.name
    def tell(self):
        '''Tell my details.'''
        print 'Name:"%s" Age:"%s"'% (self.name, self.age)
class Teacher(SchoolMember):
    '''Represents a teacher.'''
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print'(Initialized Teacher: %s)'% self.name
    def tell(self):
        SchoolMember.tell(self)
        print'Salary: "%d"'% self.salary
```

```
t = Teacher("lgl",28,1000)
t.tell()
```

```
(Initialized SchoolMember: lgl)
(Initialized Teacher: lgl)
Name:"lgl" Age:"28"
Salary: "1000"
```

Python文件读写

文件

- 打开文件
  - ✓ f = file(filename, 'w')
    - w 写 r 读 a 追加
    - 关闭文件
  - ✓ f.close ()
    - 当前位置
  - ✓ f.tell ()
    - 寻址
  - ✓ f.seek (offset, position)
    - position = 0 文件开头为原点
    - position =1 当前位置为原点
    - position =2 文件结尾为原点

读写文件

- 读取文件
  - ✓读取一行 **f.readline()**
  - ✓读取多行 **f.readlines()** 返回一个**list**
  - ✓读取指定长度 **f.read(100)**
  - ✓读取全部 **f.read()**
- 写文件
  - ✓写入一个值 **f.write (value)**
  - ✓写一行 **f.writeline(signle-line)**
  - ✓写多行 **f.writelines(multi-lines)**

读写文件例子

```
spath=“test.txt”
f=open(spath,"w")
f.write(“line 1.\n")
f.writelines(“line 2.\n \line 3 \n")
f.close()
f=open(spath,"r")
for line in f:
    print line
f.close()
```

异常

- try except
  - ✓try:
    - ……..
  - ✓except:
    - ……..
- try finally
  - ✓Finally 模块里语句必须执行
  - ✓用于释放资源

异常例子

```
import sys
try:
    s = raw_input('Enter something --> ')
except EOFError:
    print '\nWhy did you do an EOF on me?'
except:
    print '\nSome error/exception occurred.'
print 'Done'
```

dir函数

显示数据类型支持的函数

```
>>> l = [1,2,3]
>>> dir(l)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
    '__delslice__', '__doc__', '__eq__', '__format__', '__ge__',
    '__getattribute__', '__getitem__', '__getslice__', '__gt__',
    '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__',
    '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
    '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
    '__setattr__', '__setitem__', '__setslice__', '__sizeof__', '__str__',
    '__subclasshook__', 'append', 'count', 'extend', 'index', 'insert',
    'pop', 'remove', 'reverse', 'sort']
```

help模块

- 系统帮助
  - ✓help(str)
  - ✓help(list)
- help(list)

```
Help on class list in module __builtin__:
class list(object)
 list() -> new list
 list(sequence) -> new list initialized from sequence's items
Methods defined here:
__add__(...)
    x.__add__(y) <==> x+y
```

sys模块

- 参数sys.argv
- 平台 sys.platform
- 版本 sys.version
- 输入输出 sys.stdin sys.stdout

sys.argv

- 获得参数列表
- 脚本的名称总是sys.argv列表的第一个参数

```
import sys
for i in sys.argv:
    print i
```

- python *using_sys.py* we are arguments
  - ✓ argv[0] = *using_sys.py*
  - ✓ argv[1] = we
  - ✓ argv[2] = are
  - ✓ argv[3] = arguments

sys.argv

- python *using sys.py* “we are arguments”
  - ✓argv[0] = *using sys.py*
  - ✓argv[1] = “we are arguments”
- python *using sys.py* ‘we are arguments’
  - ✓argv[0] = *using sys.py*
  - ✓argv[1] = ‘we
  - ✓argv[2] =are
  - ✓argv[3]=arguments’

sys.version

- Python 版本信息
- sys.version
  - '2.6.2 (r262:71605, Apr 14 2009, 22:40:02) [MSC v.1500 32 bit (Intel)]'
- sys.version_info （以tuple的形式返回）
  - (2, 6, 2, 'final', 0)

其他

- 平台信息：**sys.platform**
  - ✓ **'Win32'**
- 环境变量：**sys.path**
  - ✓ **['C:\\Python26\\Lib\\idlelib', 'C:\\Windows\\system32\\python26.zip', 'C:\\Python26\\DLLs', 'C:\\Python26\\lib', 'C:\\Python26\\lib\\plat-win', 'C:\\Python26\\lib\\lib-tk', 'C:\\Python26', 'C:\\Python26\\lib\\site-packages', 'C:\\Python26\\lib\\site-packages\\win32', 'C:\\Python26\\lib\\site-packages\\win32\\lib', 'C:\\Python26\\lib\\site-packages\\Pythonwin']**
- 输入输出：**sys.stdout sys.stdin sys.stderr**
- 终止：**sys.exit(0)**

os 模块

- ✓ os.name 平台
- ✓ os.getcwd() 工作目录
- ✓ os.getenv('path')、os.putenv() 环境变量
- ✓ os.listdir('path') 显示目录
- ✓ os.mkdir ('path') 创建目录
- ✓ os.rmdir ('path') 删除目录
- ✓ os.curdir:返回当前目录（'.')
- ✓ os.chdir(dirname):改变工作目录到dirname
- ✓ os.remove('path') 删除
- ✓ os.linesep 平台的行终止符

time模块

- import time
- time.time() 1281282117.892
- time.localtime()
  - time.struct_time(tm_year=2010, tm_mon=8, tm_mday=8, tm_hour=23, tm_min=42, tm_sec=31, tm_wday=6, tm_yday=220, tm_isdst=0)
- time.asctime()
  - 'Mon Sep 05 18:44:29 2011'

time模块

- time.strftime('%Y-%m-%d %H:%M:%S', time.gmtime())
  - ✓ **2010-08-08 15:45:28**

%y 两位数的年份表示(00-99)
%Y 四位数的年份表示(0000-9999)
%m 月份（01-12）
%d 月内中的一天（0-31）
%H 24小时制小时数（0-23）
%I 12小时制小时数（01-12）
%M 分钟数（00-59）
%S 秒（00-59）

%a 本地简化星期名称
%A 本地完整星期名称
%b 本地简化的月份名称
%B 本地完整的月份名称
%c 本地相应的日期表示和时间表示
%j 年内的一天（001-366）
%p 本地A.M.或P.M.的等价符
%U 一年中的星期数（00-53）星期天为星期的开始
%w 星期（0-6），星期天为星期的开始
%W 一年中的星期数（00-53）星期一为星期的开始
%x 本地相应的日期表示
%X 本地相应的时间表示
%Z 当前时区的名称

Math 模块

```
from math import *


print e, pi
print cos(radians(180.0))
print log(10.0)
print exp(-1.0)
```

Math的方法

函数（方法）	示例	说明
acos(x)	求x的反余弦（结果是弧度）	acos(2.0)等于0.0
asin(x)	求x的反正弦（结果是弧度）	asin(0.0)等于0.0
atan(x)	求x的反正切（结果是弧度）	atan(0.0)等于0.0
ceil(x)	为x取整，结果是不小于x的最小整数	ceil(9.2)等于10.0
cos(x)	求x的余弦（x是弧度）	cos(0.0)等于1.0
exp(x)	求幂函数e	exp(1.0)等于2.71828
fabs(x)	求x的绝对值	fabs(-5.1)等于5.1
floor(x)	为x取整，结果是不大于x的最大整数	floor(-9.8)等于-10.0
fmod(x,y)	求x/y的余数，结果是浮点数	fmod(9.8,4.0)等于1.8
hypot(x,y)	求直角三角的斜边长度，直边长度为x和y：Sqrt(x^2-y^2)	hypot(3.0,4.0)等于5.0
log10(x)	求x的对数（以10为底）	log10(10.0)等于1.0
pow(x,y)	求x的y次方（x^y）	pow(2.7,7.0)等于128.0
sin(x)	求x的正弦（x是弧度）	sin(0.0)等于0.0
sqrt(x)	求x的平方根	sqrt(900.0)等于30.0
tan(x)	求x的正切（x是弧度）	tan(0.0)等于0.0

产生一系列整数 Range

- range([start,=0] stop[, step=1])
  - ✓Start 起始值
  - ✓Stop 结束值
  - ✓Step步长
- range(0,10,2)
  - ✓[0, 2, 4, 6, 8]
- range(5,10)
  - ✓[5, 6, 7, 8, 9]
- range(10)
  - ✓[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Map

map函数作用于给定序列的每个元素，并用一个列表来提供返回值。

```
>>> from math import *
>>> r = [0, 1, 2, 3, 4, 5, 6]
>>> map(cos, r)
[1.0, 0.54030230586813977, -
0.41614683654714241, -
0.98999249660044542,
 -0.65364362086361194,
0.28366218546322625, 0.96017028665036597]
```

reduce

reduce函数为二元函数，作用于序列的元素，每次携带一对（先前的结果以及下一个序列的元素），连续的将现有的结果和下一个值作用在获得的随后的结果上，最后减少我们的序列为一个单一的返回值。

```
>>> r = [0, 1, 2, 3, 4, 5, 6]
>>> def sum(x, y): return x+y
>>> reduce(sum, r)
# ((((((1+2)+3)+4)+5)+6)
```

```
21
```

求解1000的阶乘

```
print reduce(lambda x,y:x*y, range(1, 1001))
```

filter 过滤

filter函数的功能相当于过滤器。

```
>>> r = [0, 1, 2, 3, 4, 5, 6]
>>> def large(x): return x>3
>>> filter(large, r)
```

```
[4, 5, 6]
```

交叉合并

- zip(range(2,10), ('a', 'b', 'c', 'd', 'e'))
- [(2, 'a'), (3, 'b'), (4, 'c'), (5, 'd'), (6, 'e')]

网页处理urllib2

- 下载网页

```
import urllib2
response = urllib2.urlopen('http://python.org/')
html = response.read()
```

- 基于请求应答的方法

```
import urllib2
req = urllib2.Request('http://www.tinoweb.cn')
response = urllib2.urlopen(req)
the_page = response.read()
```

通过Request打开的好处是，我们可以很方便的为Request 添加HTTP 请求的头部信息。

发送数据

```
import urllib
import urllib2
url = 'http://dict.youdao.com/search'
user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
headers = { 'User-Agent' : user_agent }
values = {'q' : 'python'}
data = urllib.urlencode(values)
req = urllib2.Request(url, data, headers)
response = urllib2.urlopen(req)
the_page = response.read()
print the_page
```

异常处理

```
from urllib2 import Request, urlopen, URLError, HTTPError
req = Request(someurl)
try:
    response = urlopen(req)
except HTTPError, e:
    print  'The server couldn\'t fulfill the request.'
    print 'Error code: ', e.code
except URLError, e:
    print 'We failed to reach a server.'
    print 'Reason: ', e.reason
else:
    print “everything is fine “
```

HTML解析

- **HTMLParser**
- 自己用正则表达式（最后一堂课详细介绍）

处理中文

- #coding=gbk
- s = "中文"
- print s

- #coding=utf-8
- s = u'中文' #unicode编码的文字
- print s.encode('gbk')　#转换成utf-8格式输出

Thanks
Questions?