

# Python 语言程序设计



# 第二章

## Python语言概述



Python的产生与特性 01

02 安装与运行

基本语法 03

04 程序设计基础

turtle绘图 05



# 01 Python的产生与特性

# 程序员们别打了：十大编程语言排行榜发布，Python再度夺冠



环球扫描

18-08-04 17:29



国际电子和电气工程师联合会最近发布了第五届年度编程语言排行榜，Python不但再度名列榜首，还拉开了与第二名的距离。汇编语言则首次进入前十名。

本次排行榜是通过公众APP投票进行的，共有47种语言候选。投票之后，组织方还要根据IEEE会员的意见加权。最后折算成百分制，第一名为100分，其他名次递减。



Rank	Language	Type	Score
1	Python	  	100.0
2	Java	  	96.3
3	C	  	94.4
4	C++	  	87.5
5	R		81.5
6	JavaScript		79.4
7	C#	   	74.5
8	Matlab		70.6
9	Swift	 	69.1
10	Go	 	68.0

Python 稳居榜首，且连续夺冠三年





# 世界编程语言排行榜





[编辑](#)[讨论](#)

**TIOBE**编程语言排行榜是编程语言流行趋势的一个指标，每月更新，这份排行榜排名基于互联网有经验的程序员、课程和第三方厂商的数量。排名使用著名的搜索引擎（诸如Google、MSN、Yahoo!、Wikipedia、YouTube以及Baidu等）进行计算。请注意这个排行榜只是反映某个编程语言的热门程度，并不能说明一门编程语言好不好，或者一门语言所编写的代码数量多少。

这个排行榜可以用来考查你的编程技能是否与时俱进，也可以在开发新系统时作为一个语言选择依据。



Jan 2021	Jan 2020	Change	Programming Language	Ratings	Change
1	2	⬆	C	17.38%	+1.61%
2	1	⬇	Java	11.96%	-4.93%
3	3		Python	11.72%	+2.01%
4	4		C++	7.56%	+1.99%
5	5		C#	3.95%	-1.40%
6	6		Visual Basic	3.84%	-1.44%
7	7		JavaScript	2.20%	-0.25%
8	8		PHP	1.99%	-0.41%
9	18	⬆	R	1.90%	+1.10%
10	23	⬆	Groovy	1.84%	+1.23%



# 从计算机到编程

## 程序语言的演变

- 编程其实就是把人类的需求用计算机语言来表达，是一场人与计算机的对话。
- 计算机语言经历了从机器语言、汇编语言，再到高级语言的演变过程。

## 高级语言的运行机制

- 高级语言按照执行方式可以分为编译型和解释型两种。
  - 编译程序对源程序进行解释的方法相当于日常生活中的“整文翻译”。
  - 解释程序对源程序进行翻译的方法相当于日常生活中的“同声传译”。

# 从计算机到编程

编译型语言具有如下优点：

- 可独立运行，源代码经过编译形成的目标程序可脱离开发环境独立运行；
- 运行效率高，编译过程包含程序的优化过程，编译的机器码运行效率较高。

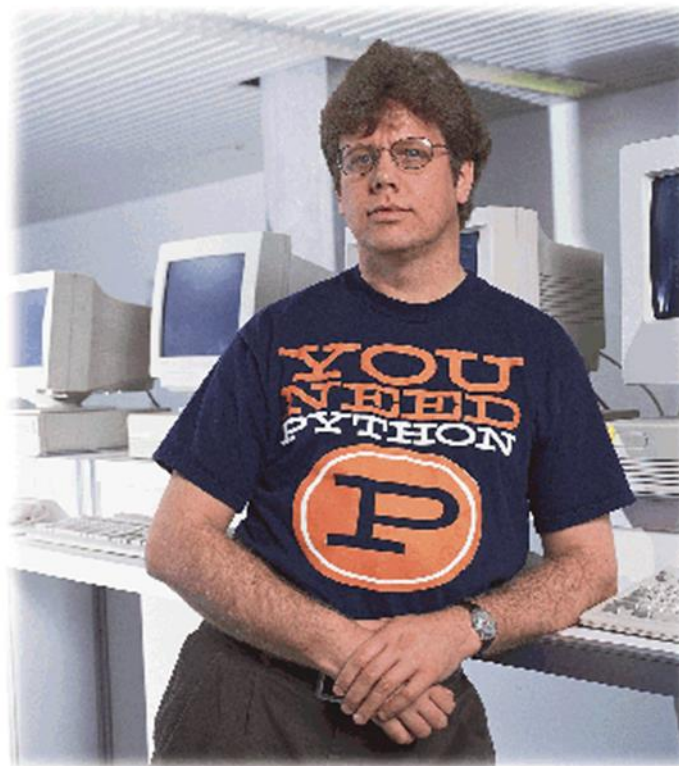
解释型语言的优点如下：

- 易于修改和测试，逐句解释过程中便于对代码的修改和测试；
- 可移植性较好，只要有解释环境，可在不同的操作系统上运行。

# Python的产生与特性

## Python语言的发展

- Python 2.0 - 2001/06/22
- Python 2.4 - 2004/11/30
- Python 2.5 - 2006/09/19
- Python 2.6 - 2008/10/01
- Python 2.7 - 2010/07/03
- Python 3.0 - 2008/12/03
- Python 3.1 - 2009/06/27
- Python 3.2 - 2011/02/20
- Python 3.3 - 2012/09/29
- Python 3.4 - 2014/03/16
- Python 3.5 - 2015/09/13
- Python 3.6 - 2016/12/23
- .....
- Python 3.9 - 2020-10-05



# 02 Python的安装与运行

# Python的产生与特性

## Python语言的特性

- 1. 语法简单
- 2. 可移植性
- 3. 粘性扩展
- 4. 开源理念
- 5. 面向对象



# Python的安装与运行

## Python的下载和安装

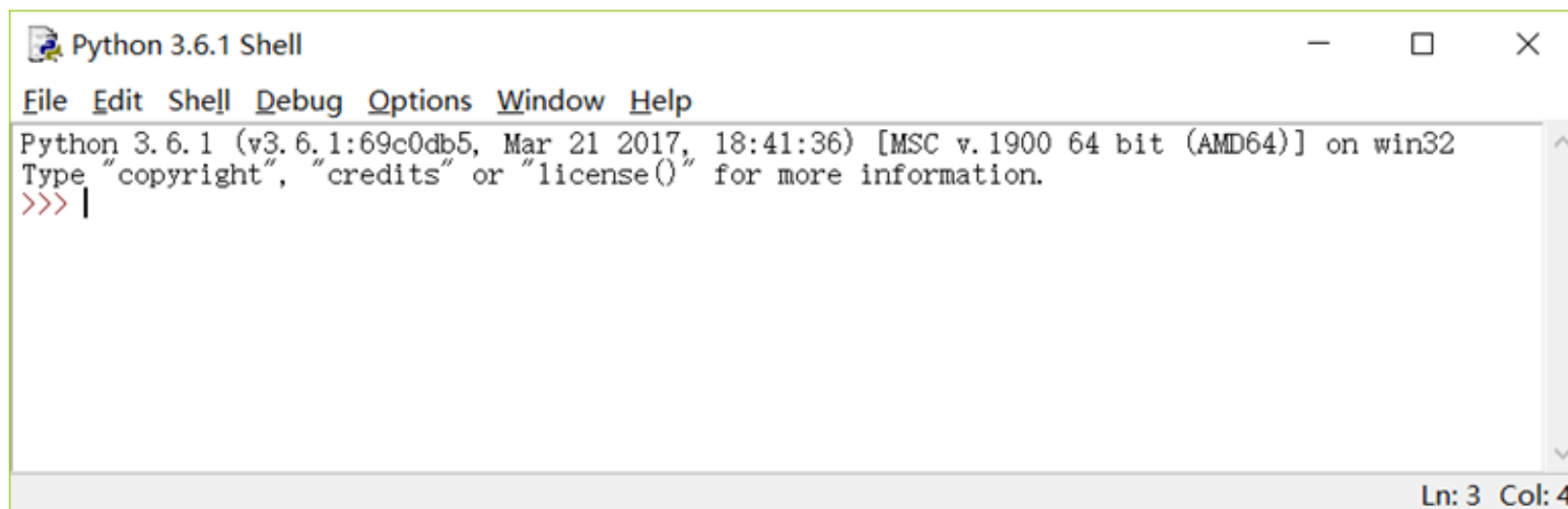
<http://www.python.org/downloads/>



# Python的安装与运行

## Python的运行

- Python安装完成后，在Windows的“开始”菜单中选择，“程序” - “Python 3.5” - “IDLE (Python 3.5 64bit)”，可以启动内置的解释器（IDLE集成开发环境）。



```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 18:41:36) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

Ln: 3 Col: 4

# Python的安装与运行

## Python的运行

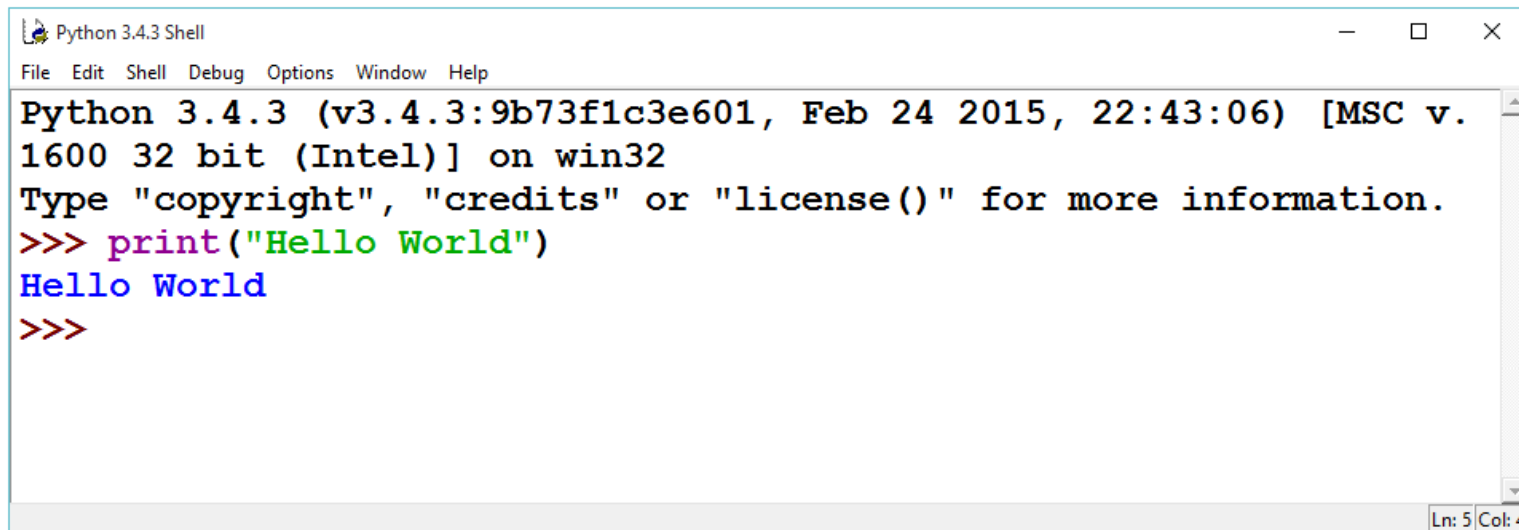
### 1. 命令行方式

- 命令行方式是一种交互式的命令解释方式
- 输入命令，解释器（Shell）即负责解释并执行命令

### 2. 文件执行方式

- 建立程序文件，然后调用并执行这个文件
- 以.py为扩展名

## ■ 命令行：调用IDLE来启动Python图形化运行环境



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.
1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello World")
Hello World
>>>
```

The screenshot shows a terminal window titled "Python 3.4.3 Shell". It displays the Python version and build information, followed by a prompt to type "copyright", "credits", or "license()". The user has entered the command `print("Hello World")`, and the output `Hello World` is displayed. The prompt `>>>` is shown again at the bottom. The status bar at the bottom right indicates "Ln: 5 | Col: 4".

## ■ 文件：Ctrl+N新建，输入语句并保存（.py），使用快捷键F5即可运行该程序。

# 03 Python的基本语法

# Python的基本语法

## 初识程序

【例2.1】绘制基本图形：使用turtle库绘制正N边形。

#example2.1这是一个绘制正N边形的程序

```
import turtle as t
```

```
n=eval(input('请输入绘制的边数N:'))
```

```
w=eval(input('请输入绘制的线宽W:'))
```

```
t.reset()
```

```
t.pensize(w)
```

```
t.circle(100,None,steps=n)
```

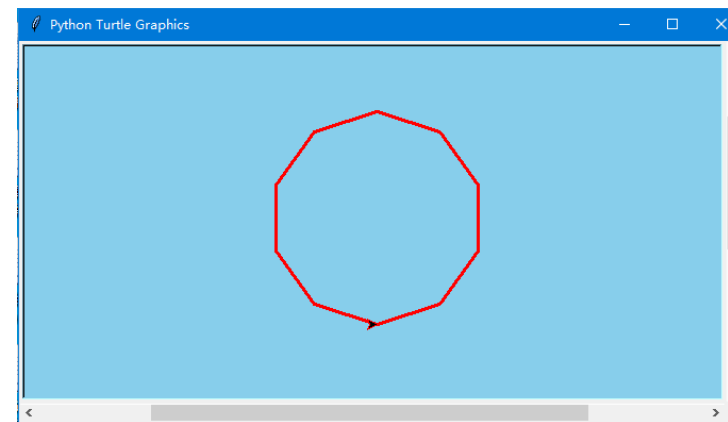
```
>>>
```

```
===== RESTART:C:\Users\Python\example2.1.py =====
```

```
请输入绘制的边数N:10
```

```
请输入绘制的线宽W:3
```

```
>>>
```



# Python的基本语法

## 初识程序

【例2.2】简单计算问题： 100以内自然数的累加和

#example 2.2

```
sum=0
```

```
for i in range(1,101):
```

```
    sum=sum+i
```

```
print("自然数1-100的累加和为： ",sum)
```

程序运行结果如下：

```
>>>
```

```
==== RESTART: C:\Users\Python\example2.2.py ====
```

```
自然数1-100的累加和为： 5050
```

```
>>>
```

# Python的基本语法

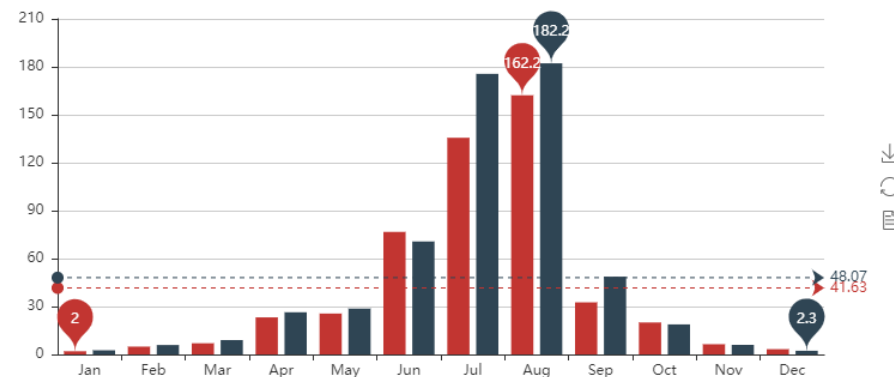
## 初识程序

【例2.3】数据可视化：用Pyecharts库绘制柱形图。

```
1 #example2.3
2 from pyecharts import Bar
3 columns = ["Jan", "Feb", "Mar", "Apr", "May", "Jun"]
4 data1 = [2.0, 4.9, 7.0, 23.2, 25.6, 76.7]
5 data2 = [2.6, 5.9, 9.0, 26.4, 28.7, 70.7]
6 bar = Bar("柱状图", "2020年上半年降水量与蒸发量")
7 bar.add("降水量", columns, data1, \
        mark_line=["average"], mark_point=["max", "min"])
8 bar.add("蒸发量", columns, data2, \
        mark_line=["average"], mark_point=["max", "min"])
9 bar.render("chart1.html")
```

柱状图

一年的降水量与蒸发量



# Python的基本语法

## 语法规则

- 注释：用来在程序中对语句、运算等进行说明和备注。适当的添加注释语句可以很好地增加程序的可读性，同时也便于代码的调试和纠错。
  - “#”符号表示单行注释，用 “''' ”（3个单引号）表示多行注释。
  - 仅是说明性文字，不会作为代码而被执行。
  - 在IDLE窗口中，以红色或绿色文字标出，用以区别代码部分。
- 关键字（Keyword）又称保留字，是Python系统内部定义和使用的特定标识符。Python 3.5.X中共有33个关键字。

# Python的基本语法

## 语法规则

- 标识符：标识符用来表示常量、变量、函数、对象等程序要素的名字。
  - 可以使用任何能够明确说明该数据特征、用途、性质的字符
  - 可以是包括由字母、数字、下划线或汉字组成的字符串。
  - 必须要符合下面的命名规则：
    - (1) 首字符必须是字母、汉字或下划线。
    - (2) 中间可以是字母、汉字、下划线或数字，但不能有空格。
    - (3) 字母区分大小写（大写S和小写s代表了不同的两个名称）。
    - (4) 不能使用Python的关键字。

# Python的基本语法

## 语法规则

- 强制缩进：使用缩进来表示代码块，不需要使用大括号“{}”。
- 缩进的空格数是可变的，但是同一个代码块的语句必须包含相同的缩进空格数。

```
#example 修改1
sum=0
for i in range(1,101):
    sum=sum+i
    print("自然数1-100的累加和为：",sum)
```

```
#example 修改2sum=0
for i in range(1,101):
    sum=sum+i
    print("自然数1-100的累加和为：",sum)
```

# Python的基本语法

- 语法规则

- 一句多行：Python 通常是一行写完一条语句，但如果语句过长，可以使用反斜杠 “\” 来实现多行语句。

```
7 bar.add("降水量", columns, data1, \
    mark_line=["average"], mark_point=["max", "min"])
8 bar.add("蒸发量", columns, data2, \
    mark_line=["average"], mark_point=["max", "min"])
```

# Python的基本语法

- 语法规则

- 多句一行：Python可以在同一行中使用多条语句，语句之间使用分号 “;” 分割。
- 例如，在IDLE命令行输入一行语句，会执行三条命令，分别是半径r等于10、计算面积s、显示计算结果。
- 语句的输入及运行结果如下：

```
>>> r=10;s=3.1415926*r*r;print("圆的面积为：",s)
圆的面积为： 314.15926
>>>
```

# 04 程序设计基础

# 问题求解的程序结构-IPO

【例2.4】欧几里得算法（求最大公约数）的程序实现。

# example2.4

```
a,b=eval(input('输入两个自然数a, b: '))
```

```
r=a%b
```

```
while r!=0:
```

```
    a=b
```

```
    b=r
```

```
    r=a%b
```

```
print('最大公约数为: ',b)
```

数据输入（Input）

数据处理（Process）

数据输出（Output）

程序运行结果如下：

```
>>>
```

```
===== RESTART: C:\Users\example2.4.py =====
```

```
输入两个自然数a,b==>12075,4655
```

```
最大公约数为: 35
```

```
>>>
```

# 问题求解的程序结构-IPO

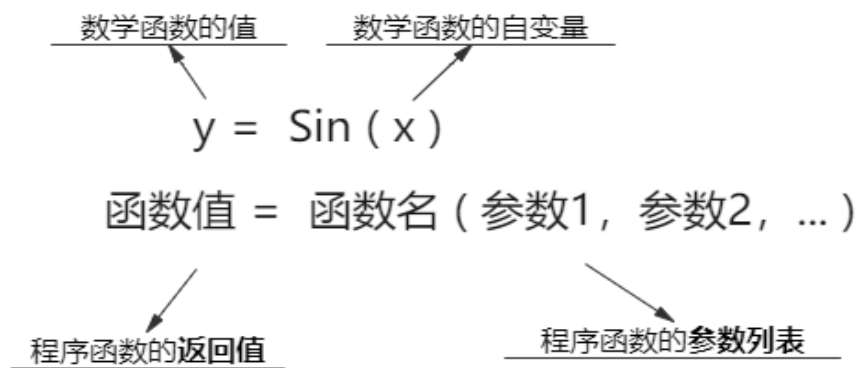
【例2.5】求一元二次方程 $ax^2+bx+c=0$ 的实根。

```
# example2.5
a=int(input("请输入a: ")) #输入a
b=int(input("请输入b: ")) #输入b
c=int(input("请输入c: ")) #输入c
d=b**2-4*a*c
if d>=0:                                #判断是否有实根
    x1=(-b+d**0.5)/(2*a)
    x2=(-b-d**0.5)/(2*a)
    print('方程的根: x1=%f,x2=%f'%(x1,x2))
else:
    print('input data Error! ')
```

# 函数是什么？

高级语言中的**函数**就是一段代码的**封装**，用来实现某种特定的运算或功能。

程序中函数的**调用**与数学函数中方法类似，都是要通过**函数名**并且给定**参数**的方式。



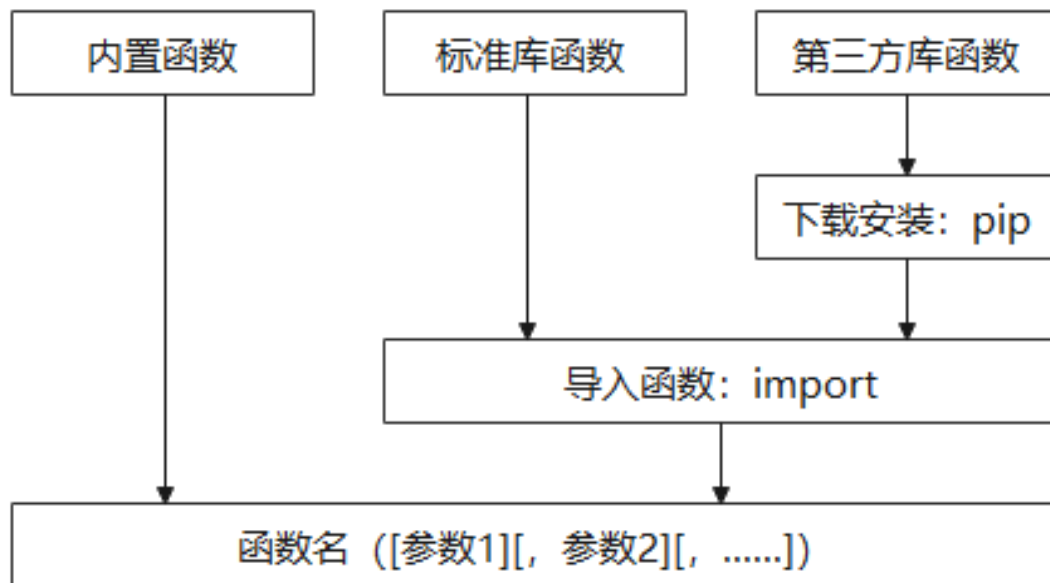
# Python的函数有三种

**内置函数**是系统自带的可以直接使用的函数，如，input、print、eval等。

**标准库函数**是系统自带的外部函数，需要先用import导入函数库后才能使用，  
如，turtle是标准函数库。

**第三方库函数**由相关人员或机构开发，需要先进行函数库的安装，再导入后才能使用。如，pyecharts。

# Python的函数有三种



# 输入函数input()

## 函数的语法格式

input 是输入函数，用来读取用户输入的数据，并返回一个字符串，具体格式如下：

```
<变量名>=input(["提示信息"])
```

变量：用来存放  
输入的数据

提示信息：是一个字符串，要放在  
英文的引号当中。也可以省略不写

```
x = input("请输入数据：")
```

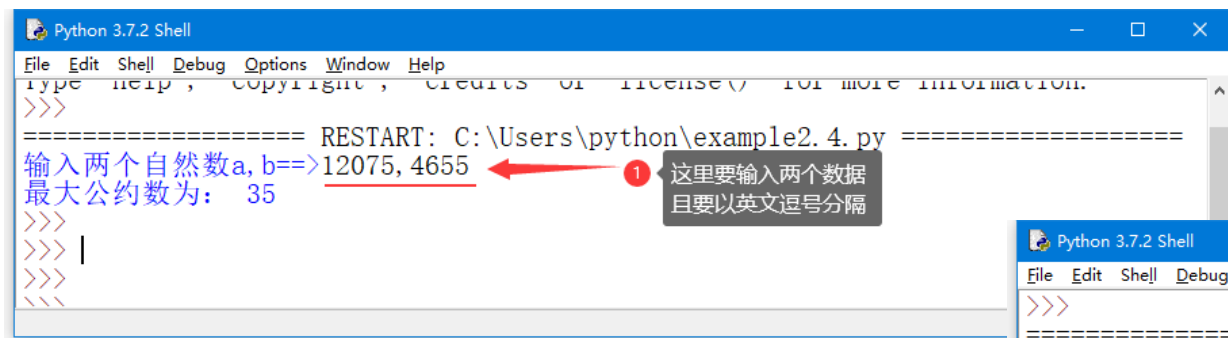
赋值：将右边的数据赋给左边的变量

# 输入函数input()

## 为多个变量赋值

例如：在【例 2.4】中，输入两个自然数的语句：

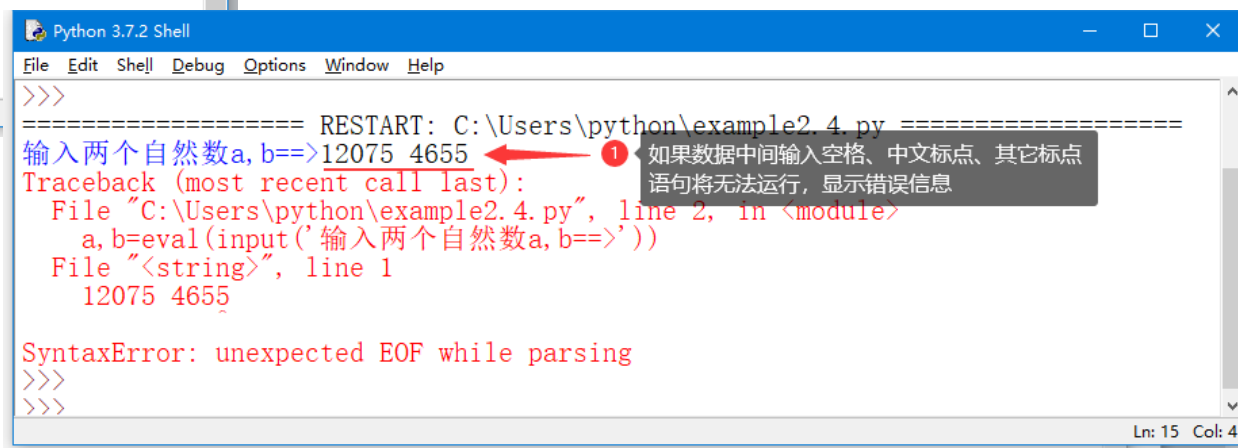
```
a,b=eval(input('输入两个自然数 a, b: '))
```



Python 3.7.2 Shell

```
>>>
===== RESTART: C:\Users\python\example2.4.py =====
输入两个自然数a,b==>12075,4655
最大公约数为: 35
>>>
>>> |
>>>
```

这里要输入两个数据  
且要以英文逗号分隔



Python 3.7.2 Shell

```
>>>
===== RESTART: C:\Users\python\example2.4.py =====
输入两个自然数a,b==>12075 4655
Traceback (most recent call last):
  File "C:\Users\python\example2.4.py", line 2, in <module>
    a,b=eval(input('输入两个自然数a,b==>'))
  File "<string>", line 1
    12075 4655
SyntaxError: unexpected EOF while parsing
>>>
>>>
```

如果数据中间输入空格、中文标点、其它标点  
语句将无法运行，显示错误信息

Ln: 15 Col: 4

# 输入函数input()

## 使用函数转换数据类型

input函数在默认情况，返回的值是输入数据的字符串。也就是说，函数会将完整的用户输入形成一个字符串返回。

当需要使用input函数返回数值型数据时，则须使用类型转换函数来进行转换。下面两条语句，分别将输入数据转换为整数类型和原数据类型

int函数：用来将输入的字符串转换为整数

`x = int (input ( " 请输入数据：" ))`

`x = eval (input ( " 请输入数据：" ))`

eval函数：用来将输入的字符串转换为原数据

# 输入函数input()

## 使用函数转换数据类型

```
>>> x=input('input x=>')
input x=>5
>>> y=input('input x=>')
input x=>20
>>> print(x+y)
520
>>>
```

```
>>> x=int(input('input x=>'))
input x=>5
>>> y=val(input('input x=>'))
input x=>10
>>> x+y
15
>>>
```

# 输出函数print()

## 函数的语法格式

print 是输出函数，用来输出表达式的值，具体格式如下：

```
print(value1, value12,...,sep=' ',end='\n')
```

### 说明：

- (1) value：表示要输出显示的值的列表，各个值之间用 “,”（逗号隔开）。print输出显示值列表中常量变量的值，以及其中表达式的计算结果。
- (2) sep=' '：表示在各输出值间的分隔符，默认情况下以空格分隔。修改sep的值可以自定义分隔符。
- (3) end='\n'：表示添加到最后一个输出值后面的结束符，默认情况下以回车结束（即换行输出）。修改end的值可以自定义结束符。
- (4) 无参数时，print函数输出一个空行。

# 输出函数print()

【例2.6】print的测试程序。

```
#example2.6
print("~~~~~这是一个print的小测试~~~~~")
print()                #输出空行
print("Python Python",5,20)
print("Python"+"Python",5*100+20)
a="Python";b=8*100+80;c=51    #给三个变量赋值
print(a*3,b,sep="51")
print(a*3,c,b,sep="~ ",end="hehehe!")
print()                #输出空行
print(c,b,sep="",end="")
print(c+40,b,sep="")
```

程序运行结果如下：

```
>>>
===== RESTART: C:\Users\example2.6.py =====
~~~~~这是一个print的小测试~~~~~

Python Python 5 20
PythonPython 520
PythonPythonPython51880
PythonPythonPython~51~880hehehe!
5188091880
>>>
```

# 输出函数print()

## 控制数据输出的样式

### (1) 分行显示字符串

【例2.7】两种字符串分行显示的方法示例。

#example2.7 荀子之劝学

劝学=" 劝 学\n 荀子"

劝学1="""

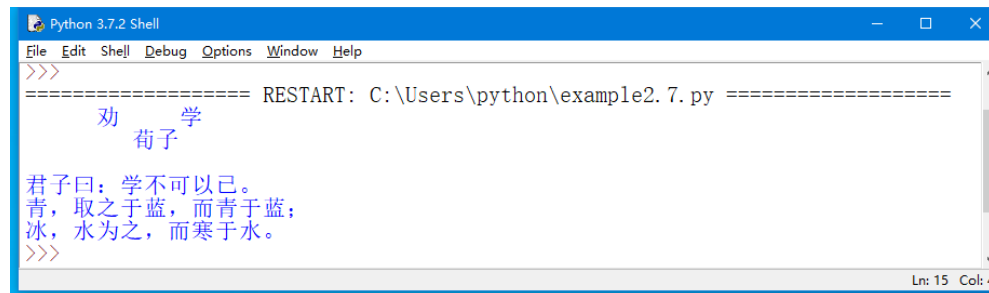
君子曰：学不可以已。

青，取之于蓝，而青于蓝；

冰，水为之，而寒于水。"""

print(劝学)

print(劝学1)



```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
>>>
===== RESTART: C:\Users\python\example2.7.py =====
劝学
荀子
君子曰：学不可以已。
青，取之于蓝，而青于蓝；
冰，水为之，而寒于水。
>>>
Ln: 15 Col: 4
```

程序中用到了两种字符串分行的方法：

第一种方法，在字符串加入转义字符“\n”，print时会将其转义为一个“回车符”。

第二种方法，使用三引号“"""”定义一个多行字符串，print会原样输出此字符串。

# 输出函数print()

## 控制数据输出的样式

### (2) 设置数据的小数位数

【例2.8】计算圆的面积并保留4位小数。

```
#example2.8
```

```
半径=eval(input("请输入半径: "))
```

```
面积=3.1415926*半径**2
```

```
print('====计算结果====')
```

```
print('圆的面积为: %.4f'%面积)
```

运行结果如下:

```
>>>
```

```
====RESTART: C:\Users\example2.8.py ====
```

```
请输入半径: 12.4
```

```
====计算结果====
```

```
圆的面积为: 483.0513
```

```
>>>
```

格式说明符: 此处显示一个保留4位小数的浮点数

```
print( "圆的面积为: %.4f" % 面积 )
```

变量: 字符串中要显示的值, 存放计算结果

# 转换函数eval()

## 函数的语法格式

eval 函数用来执行一个字符串表达式，并返回表达式的值。具体格式如下：

```
eval(表达式[,globals[,locals]])
```

说明：

- (1) 表达式：必须是一个字符串表达式。
- (2) globals：变量作用域，全局命名空间，必须是一个字典对象，此参数为可选参数。
- (3) Locals：变量作用域，局部命名空间，可以是任何映射对象，此参数为可选参数。

将字符串引号当中的内容提取出来形成一个表达式，如果表达式合法有效，则求值并返回计算结果。如果表达式无效，则提示错误。

```
>>> x=7
>>> y=eval('3*x')
>>> z=eval('x+y')
>>> print(x,y,z)
7 21 28
>>>
```

## 转换函数eval()

【例2.9】简单公式计算器。

```
#example2.9
print("\n=====这是一个简单公式计算器=====")
print("=====公式中必须使用英文符号=====\n")
expr=eval(input("请在这里输入要计算的公式: "))
print("\n=====计算结果为:",expr)
```

程序运行结果如下：

```
>>>
=====这是一个简单公式计算器=====
=====公式中必须使用英文符号=====

请在这里输入要计算的公式: 5+2*10-(18/6)

=====计算结果为: 22.0
>>>
```

程序只用了四条语句就完成一个公式计算器，通过三个函数分别实现了输入、运算与输出。其中，input函数获取用户输入并返回字符串，eval函数提取该字符串中的表达并计算，print函数输出显示结果。

# 变量与赋值

在程序运行过程中，其值不发生改变的数据对象称为**常量**。

在程序运行过程中，可以随着程序的运行更改的量称之为**变量**。

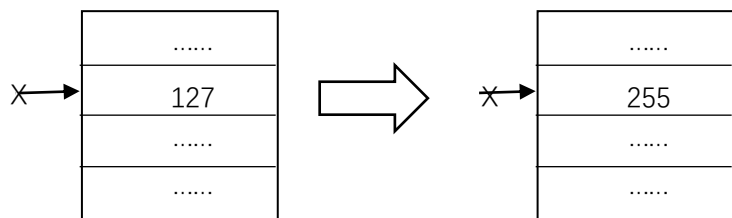
Python中的变量不需要声明，变量的**赋值**操作**即**是变量的**声明**和定义的过程。

**变量名**由字母、数字或下划线组成，变量名的首字符必须是字母或下划线，且不能为Python的关键字。

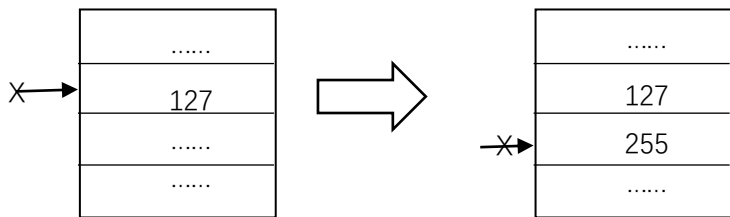
变量命名要**符合**标识符的**命名规则**。

# 变量与赋值

## 高级语言中的变量



## Python中的变量



Python语言使用赋值语句完成变量的定义。

变量名并不是对内存地址的引用，而是对数据的引用。

用赋值语句对变量重新赋值时，Python为其分配了新的内存单元，变量将指向新的地址，变量的地址发生了变化。

使用id函数可以查看变量的内存地址：

```
>>> x=127
>>> id(x)
1412875264
>>> x=255
>>> id(x)
1412879360
>>>
```

# 变量与赋值

Python变量未经赋值就使用，解译器会提示错误。

如下例中，在**变量z没有赋值**的前提下，不能直接输出z的值，否则会提示运行**错误**。  
也就是说，当给变量赋值时，就创建了该变量及数据类型。

```
>>> print(z)
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    print(z)
NameError: name 'z' is not defined
>>>
```

# 变量与赋值

## 变量的赋值

### (1) 一般形式

在Python中，赋值号就是等号“=”。赋值语句的一般形式为：

**<变量名> = <表达式>**

赋值号的左边必须是变量名，右边则是表达式。赋值语句先计算表达式的值，然后使该变量指向该数据。

```
>>> a=0                                #变量a指向数据0
>>> b=1                                #变量b指向数据1
>>> print(id(a),id(b))
1412871360 1412871456    #变量a变量b分别指向不同的数据
>>> a=b                                #变量a指向变量b数据
>>> print(id(a),id(b))
1412871456 1412871456    #变量a变量b分别指向相同的数据
>>>
```

# 变量与赋值

## 变量的赋值

### (2) 增量赋值

Python中提供了12种增量赋值运算符:

**`+=`、`-=`、`*=`、`/=`、`//=`、`%=`、`**=`、`<<=`、`>>=`、`&=`、`|=`、`^=`**

例如: 赋值语句`x+=5`相当于`x=x+5`, 如下例:

<code>&gt;&gt;&gt; x=10</code>	<code>#x赋值为5</code>
<code>&gt;&gt;&gt; x+=5</code>	<code>#相当于x=x+5</code>
<code>&gt;&gt;&gt; x</code>	<code>#x的当前值为15</code>
<code>15</code>	
<code>&gt;&gt;&gt; x*=5</code>	<code>#相当于x=x*5</code>
<code>&gt;&gt;&gt; x</code>	<code>#x的当前值为75</code>
<code>75</code>	
<code>&gt;&gt;&gt; x/=5</code>	<code>#相当于x=x/5</code>
<code>&gt;&gt;&gt; x</code>	
<code>15</code>	<code>#x的当前值为15.0</code>

# 变量与赋值

## 变量的赋值

### (3) 链式赋值

链式赋值的语句形式为：

**<变量1> = <变量2> = ..... = <变量n> = <表达式>**

链式赋值用于为多个变量赋一个相同的值。链式赋值先计算最后的表达式的值，然后将变量全部指向此数据对象。下例中x, y, z指向同一个数据对象3.1415926。

```
>>> x=y=z=3.1415926          #将3.1415926同时赋值给x,y,z
>>> print(id(x),id(y),id(z))  #x,y,z指向同一个地址
34190728 34190728 34190728
>>> print(x,y,z)
3.1415926 3.1415926 3.1415926
>>>
```

# 变量与赋值

## (4) 多重赋值

多重赋值语句的形式为：

**<变量1>, <变量2>, ....., <变量n> = <表达式1>, <表达式2>, .....<表达式n>**

赋值号两边的变量和表达式数量要一致，多重赋值首先计算赋值号右边的各表达式的值，然后按照顺序分别赋值给左边的变量。

```
>>> a,b=3,5
>>> print(a,b)           #多重赋值的结果a=3, b=5
3 5
>>> a,b=3,a
>>> print(a,b)           #多重赋值的结果a=3, b=3
3 3
>>>
```

使用多重赋值语句可以方便地实现两个变量的数据交换。例如：

```
>>> a,b=3,5
>>> a,b=b,a               #多重赋值实现a,b的数据交换
>>> print(a,b)
5 3
>>>
```

# 05 turtle绘图

## 标准库的导入

- 在Python中，**函数库**又被称为**模块**。
- 模块是一个包含所有定义的函数和变量的文件，其后缀名**.py**。
- 模块可以被其他程序引入，实现调用该模块中函数的功能，用**import** 或者 **from...import** 来导入。
- import语句既可以导入一个自定义模块，也可以导入Python标准模块。
- 函数库中的标准库和第三方库都需要先导入再调用。

## 标准库的导入

导入模块的语句是import，它有下面的三种形式：

(1) 导入一个或多个模块的全部函数，格式为：

```
import <模块名1> [,<模块名2>[,...<模块名N>] [as <别名>]
```

(2) 导入某个模块的指定函数，格式为：

```
from <模块名> import <函数名1> [,<函数名2>[,...<函数名N>]
```

(3) 导入某个模块的全部函数，格式为：

```
from <模块名> import *
```

# 标准库的导入

说明:

i. 在使用第1种形式导入模块后，在调用函数名前需要加上模块名做为前缀。如，下面的两条语句的作用是，在导入turtle库后，调用turtle.forward函数使小海龟前进15个像素。

```
>>> import turtle
>>> turtle.forward(15)
```

ii. 使用第2种方式和第3种方式导入模块后，函数名的前缀则可省略。如，下面的语句与上面的语句作用相同，只是函数名的前缀被省略了。

```
>>> from turtle import *
>>> forward(15)
```

iii. 为了增加程序的可读性，可以使用模块别名的方式来简化函数名的前缀。如，下面语句定义的模块别名为t，可以作为函数的前缀使用。

```
>>> import turtle as t
>>> t.forward(15)
```

# 标准库的导入

【例2.10】绘制一个正方形。

```
#example2.10
```

```
import turtle as t
```

```
t.setup(300,200)
```

```
for i in range(4):
```

```
    t.forward(50)
```

```
    t.left(90)
```

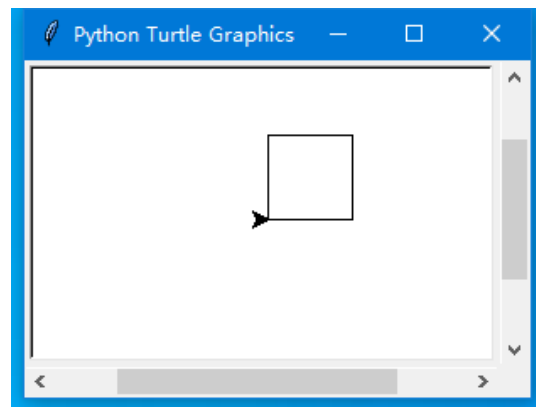
#导入turtle，别名为t

#设置画布大小

#从原点开始绘制一个正方形

#前进50个像素

#向左旋转90度



说明：程序控制海龟在300\*200像素的画布中，从画布的中心开始绘制一个边长为50像素的正方形。程序中的for语句是循环结构，经过4次循环分别绘制正方形的四条边。



# 窗口与画布

## 1. 绘图窗口

### (1) 设置窗口

绘图窗口（Python Turtle Graphics）是turtle绘图的主窗口，默认情况下，窗口宽度为当前屏幕宽度的50%，高度为当前屏幕高度的75%，位置在屏幕中心。也就是说，默认的绘图窗口的大小会根据当前使用的电脑屏幕分辨率而各有不同。使用setup函数可以设置绘图窗口的大小和位置。

**`turtle.setup(width,height,startx,starty)`**

说明：

**width:** 整数，表示以像素为单位的宽度；小数，表示宽度占屏幕宽度的比例。

**height:** 整数，表示以像素为单位的高度；小数，表示高度占屏幕高度的比例。

**startx:** 正数，表示窗口初始位置距离屏幕左边缘的像素距离；负数，表示距离右边缘，None表示窗口在屏幕水平居中。

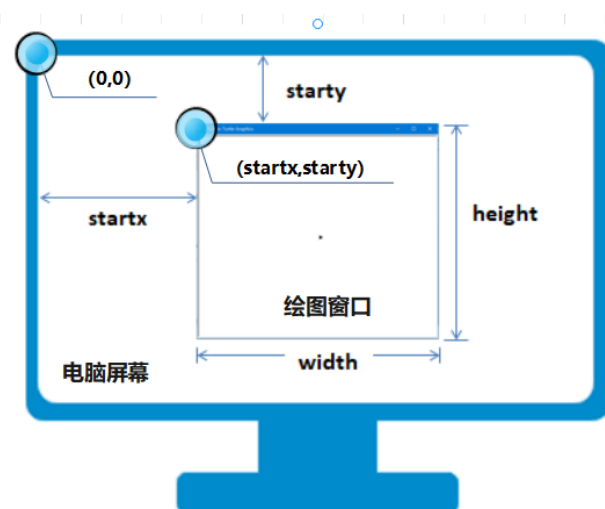
**starty:** 正数，表示窗口初始位置距离屏幕上边缘的像素距离；负数，表示距离下边缘，None表示窗口在屏幕垂直居中。

# 窗口与画布

## 1. 绘图窗口

### (2) 位置参数

setup函数的位置参数startx和starty都是指窗口左上角在当前电脑屏幕上的相对距离，绘图窗口与电脑屏幕的各参数关系如图所示。



# 窗口与画布

## 1. 绘图窗口

例如：执行下面的语句：

```
>>> import turtle
>>> turtle.setup(200, 200, 0, 0)      # 设置窗口大小为200x200像素, 初始位置在屏幕的最左上角
>>> turtle.setup(.75, 0.5, None, None) # 窗口宽度和高度为屏幕宽度和高度75%和50%, 位置居中
>>> turtle.setup()                    # 当参数都省略时, 表示设置窗口为默认的初始状态
```

### 说明：

第一条setup语句：设置窗口大小为200\*200，位置是窗口的左上角在（0，0）的位置，也就是在屏幕的左上角；

第二条setup语句：设置窗口宽度为当前屏幕宽度的75%，高度为当前屏幕高度的50%，参数startx和starty为None（可省略不写），则位置位于当前屏幕中心，此设置为系统默认设置。

第三条setup语句：当所有参数都省略时，将设置窗口大小和位置为默认的初始状态。

绘图窗口会在海龟发生绘图动作时，自动弹出。画布位于绘图窗口中，海龟在画布中爬行形成了图形。当绘图窗口小于画布时，窗口两侧会出现滚动条。

# 窗口与画布

## 2. 设置画布

画布就是turtle的绘图区域。默认情况下，画布的大小为400\*300，即，画布的宽度为400像素，高度为300像素，画布位于窗口中心。可以使用screensize函数设置它的大小和背景颜色。

**`turtle.screensize(canvwidth=None, canvheight=None, bg=None)`**

说明：

canvwidth：正整数，表示画布的像素宽度。

canvheight：正整数，表示画布的像素高度。

bg：颜色字符串或颜色元组，表示画布的背景颜色。



# 窗口与画布

## 2. 设置画布

例如：执行下面的语句：

```
>>> turtle.screensize()  
(400, 300)  
>>> turtle.bgcolor()  
'white'  
>>> turtle.screensize(800,600,"blue")  
>>> turtle.screensize()  
(800, 600)  
>>> turtle.bgcolor()  
'blue'  
>>>
```

说明：

`turtle.screensize()`：当参数都省略时，则返回当前画布的宽度和高度（`canvwidth`，`canvheight`）。

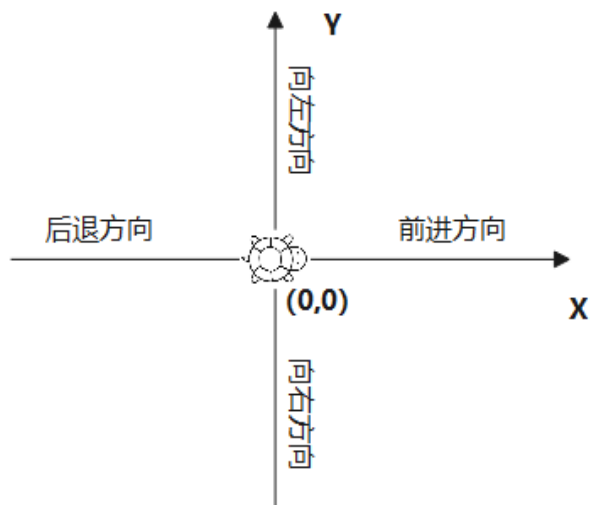
`turtle.bgcolor()`：函数用来返回当前画布的背景颜色字符串



# 窗口与画布

## 3. 坐标系统

在绘图窗口上，默认有一个坐标原点为窗口中心的坐标轴，坐标原点上有一只头朝向x轴正方向的小海龟。当海龟处于图状态时，只要控制好海龟的位置和方向，就可以在坐标系统中准确地绘制图形。窗口的坐标系统如图所示。



```
>>> import turtle as t
>>> t.pos()
(0.00,0.00)
>>> t.forward(150)
>>> t.pos()
(150.00,0.00)
>>> t.left(90)
>>> t.forward(150)
>>> t.pos()
(150.00,150.00)
>>> t.home()
>>> t.pos()
(0.00,0.00)
>>> t.goto(200,200)
>>> t.pos()
(200.00,200.00)
>>>
```

#海龟的初始位置在坐标原点(0,0)

#沿初始的前进方向移动150像素

#海龟的当前位置坐标为(150,0)

#控制海龟向左转动90度

#沿当前的前进方向移动150像素

#海龟的初始位置坐标为(150,150)

#海龟回到初始位置(0,0)和方向(x轴正方向)

#海龟移动到坐标点(200,200)



# 绘图动作与状态

## 1. 绘图状态与控制

**`turtle.pendown()` | `turtle.pd()` | `turtle.down()`**

设置画笔为可绘图状态，此时画笔放下，移动时可绘图。

**`turtle.penup()` | `turtle.pu()` | `turtle.up()`**

设置画笔为非绘图状态，此时画笔抬起，移动时不绘图。

**`turtle.pensize(width=None)` | `turtle.width(width=None)`**

设置画笔的宽度。参数为正数时，线条宽度为相应像素值。参数省略时，返回当前宽度。



# 绘图动作与状态

## 2. 绘图动作与方向

### (1) 相对移动

`turtle.forward(distance)` | `turtle.fd(distance)`

设置海龟在当前位置沿前进方向移动相应的像素距离，参数可为整数或浮点数。

`turtle.back(distance)` | `turtle.bk(distance)` | `turtle.backward(distance)`

设置海龟在当前位置沿后退方向移动相应的像素距离，参数可为整数或浮点数。

### (2) 绝对移动

`turtle.goto(x, y=None)`

`turtle.setpos(x, y=None)` | `turtle.setposition(x, y=None)`

将海龟移动至坐标系统中的一个绝对位置，其中，x为一个数字或一组坐标，y为一个数字或省略。



# 绘图动作与状态

## (3) 相对方向

`turtle.right(angle)` | `turtle.rt(angle)`

设置海龟沿当前方向向右旋转指定角度，参数可为整数或浮点数。

`turtle.left(angle)` | `turtle.lt(angle)`

设置海龟沿当前方向向左旋转指定角度，参数可为整数或浮点数。

## (4) 绝对方向

`turtle.setheading(to_angle)` | `turtle.seth(to_angle)`

设置海龟方向为一个绝对角度（相对x轴正方向的角度），整数或浮点数

## (5) 初始化海龟

`turtle.home()`

初始化海龟的位置和方向，海龟回到位置 (0, 0)，方向指向x轴正方向。

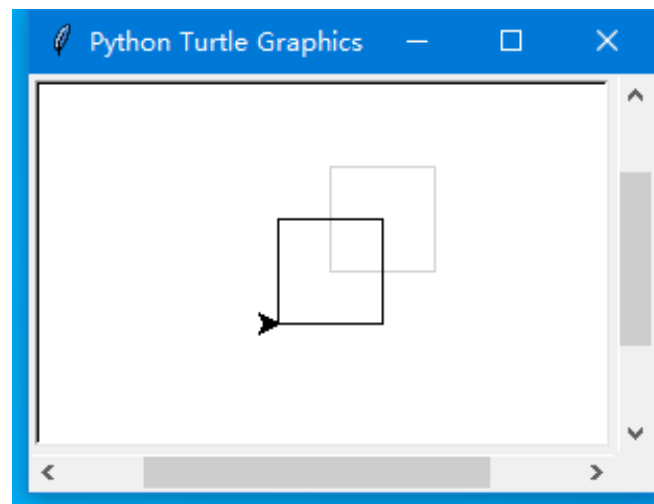


# 绘图动作与状态

【例2.11】绘制一个以原点为中心，边长为50的正方形。

#example2.11

```
import turtle as t          #导入函数库，并设置别名为t
t.setup(300,200)            #设置窗口大小为300*200
t.pensize(2)                #设置画笔线条宽度为2像素
t.penup()                   #设置画笔抬起，非绘图状态
t.goto(-25,-25)              #移动海龟至绝对位置 (-25, -25)
t.pendown()                  #设置画笔放下，进入绘图状态
for i in range(4):           #绘制正方形
    t.forward(50)
    t.left(90)
```



# 画笔控制与颜色

## 1. 画笔控制

**`turtle.circle(radius, extent=None, steps=None)`**

绘制一个半径为radius的圆。角度extent给出时，绘制一个圆弧。steps给定时，绘制圆的正内接多边形。

**`turtle.dot(size=None, *color)`**

绘制一个指定颜色和大小圆点。

**`turtle.speed(speed=None)`**

设置绘图速度。参数为0-10的一个整数，0值为最快。参数省略时，返回当前速度。

**`turtle.delay(delay=None)`**

以毫秒为单位设置动画延迟。延迟越长，动画速度越慢。参数省略时，返回当前延迟。



# 画笔控制与颜色

## 2. 颜色控制

**`turtle.color(*args)`**

设置或返回画笔颜色和填充颜色，参数省略时，返回当前画笔颜色和填充颜色。

`color(colorstring1, colorstring2)`: 参数为颜色字符串的形式。

`color((r1,g1,b1), (r2,g2,b2))`: 参数为颜色元组的形式。

用给定的第一组参数设置画笔颜色，第二组参数设置填充颜色。"

**`turtle.begin_fill()`**

形状填充的开始语句。

**`turtle.end_fill()`**

使用填充颜色填充在`begin_fill()`之后绘制的形状。

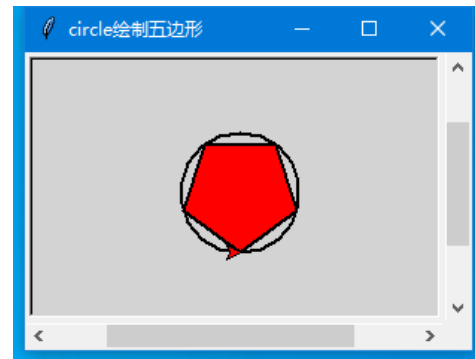


# 画笔控制与颜色

【例2.12】 绘制填充效果的多边形。

```
#example2.12
import turtle as t
t.setup(300,200,0,0)
t.title("circle绘制五边形")
t.bgcolor("light grey")
t.color("black","red")
t.dot()
t.pensize(2)
t.pu()
t.goto(0,-40)
t.pd()
t.delay(20)
t.circle(40)
t.begin_fill()
t.circle(40,steps=5)
t.end_fill()
```

#设置窗口标题  
#设置窗口背景色  
#设置画笔颜色和填充颜色  
#在原点画一个圆点



#设置延迟为20毫秒  
#绘制半径40的圆  
#开始填充  
#绘制半径40的圆的内接正五边形



分类	功能	函数名	说明
绘图状态与控制	绘图状态	<code>turtle.pendown()</code>   <code>turtle.pd()</code>   <code>turtle.down()</code>	设置画笔为可绘图状态，此时画笔移动时可绘图。
		<code>turtle.penup()</code>   <code>turtle.pu()</code>   <code>turtle.up()</code>	设置画笔为非绘图状态，此时画笔移动时不绘图。
		<code>turtle.pensize(width=None)</code>   <code>turtle.width(width=None)</code>	设置画笔的宽度。参数为正数时，线条宽度为相应像素值。参数省略时，返回当前宽度。
	当前状态	<code>turtle.position()</code>   <code>turtle.pos()</code>	返回海龟当前位置的坐标。
		<code>turtle.distance(x, y=None)</code>	返回海龟当前位置与坐标点 (x,y) 的距离。
		<code>turtle.heading()</code>	返回海龟当前的方向。



分类	功能	函数名	说明
绘图动作与方向	相对移动	turtle.forward(distance) turtle.fd(distance)	设置海龟在当前位置沿前进方向移动相应的像素距离，参数可为整数或浮点数。
		turtle.back(distance) turtle.bk(distance) turtle.backward(distance)	设置海龟在当前位置沿后退方向移动相应的像素距离，参数可为整数或浮点数。
	绝对移动	turtle.goto(x, y=None)   turtle.setpos(x, y=None)   turtle.setposition(x, y=None)	将海龟移动至坐标系中的一个绝对位置，其中，x为一个数字或一组坐标，y为一个数字或省略。
	相对方向	turtle.right(angle)   turtle.rt(angle)	设置海龟沿当前方向向右旋转指定角度，参数可为整数或浮点数。
		turtle.left(angle)   turtle.lt(angle)	设置海龟沿当前方向向左旋转指定角度，参数可为整数或浮点数。
	绝对方向	turtle.setheading(to_angle)   turtle.seth(to_angle)	设置海龟方向为一个绝对角度（相对x轴正方向的角度），整数或浮点数。
	还原默认	turtle.home()	海龟为初始状态，位置(0,0)且朝向x轴正方向。



分类	功能	函数名	说明
画笔控制与颜色	画笔控制	<code>turtle.circle(radius, extent=None, steps=None)</code>	绘制一个半径为radius的圆。角度extent给出时，绘制一个圆弧。steps给定时，绘制圆的正内接多边形。
		<code>turtle.dot(size=None, *color)</code>	绘制一个指定颜色和大小的圆点。
		<code>turtle.write(arg, move=False, align="left", font=("Arial", 8, "normal"))</code>	在当前位置书写arg中的文本。align设置文本对齐方式，font元组设置字体、字号和字形。move为True时，将画笔移动至文本右下角。
		<code>turtle.undo()</code>	
		<code>turtle.speed(speed=None)</code>	设置绘图速度。参数为0-10的一个整数，0值为最快。参数省略时，返回当前速度。
		<code>turtle.delay(delay=None)</code>	以毫秒为单位设置动画延迟。延迟越长，动画速度越慢。参数省略时，返回当前延迟。
		<code>turtle.tracer(n=None, delay=None)</code>	打开或关闭绘图动画，并设置延迟。?
	颜色控制	<code>turtle.color(*args)</code>	设置或返回画笔颜色和填充颜色，参数省略时，返回当前画笔颜色和填充颜色。 color(colorstring1, colorstring2): 参数为颜色字符串的形式。 color((r1,g1,b1), (r2,g2,b2)): 参数为颜色元组的形式。用给定的第一组参数设置画笔颜色，第二组参数设置填充颜色。"
		<code>turtle.pencolor(*args)</code>	设置画笔颜色，参数可以为颜色字符串，或者rgb颜色元组。参数省略时，返回当前颜色。
		<code>turtle.fillcolor(*args)</code>	设置或返回填充颜色。参数可以为颜色字符串，或者rgb颜色元组。参数省略时，返回当前填充颜色。
		<code>turtle.filling()</code>	返回当前颜色填充的状态。
		<code>turtle.begin_fill()</code>	形状填充的开始语句。
		<code>turtle.end_fill()</code>	使用填充颜色填充在begin_fill()之后绘制的形状。



分类	功能	函数名	说明
窗口与画布	大小位置	<code>turtle.setup(width,height,startx,starty)</code>	设置绘图窗口的宽度和高度，以及在屏幕上的位置。参数为空时，设置窗口为初始大小和位置。
		<code>turtle.screensize(canvwidth=None, canvheight=None, bg=None)</code>	设置或返回画布的大小和背景颜色。
	其它	<code>turtle.bgcolor(*args)</code>	设置绘图窗口的背景颜色，参数省略时返回当前颜色。
		<code>turtle.clear()</code>   <code>turtle.clearscreen()</code>	删除所有绘图，重置绘图窗口大小和位置为初始状态。
		<code>turtle.reset()</code>   <code>turtle.resetscreen()</code>	重置窗口中海龟为初始状态。
		<code>turtle.window_height()</code>	返回绘图窗口的高度。
		<code>turtle.window_width()</code>	返回绘图窗口的宽度。
		<code>turtle.bye()</code>	关闭绘图窗口。
		<code>turtle.title(titlestring)</code>	设置绘图窗口的标题栏。



# 小 结

- Python的产生与特性
- Python环境的安装与运行
- Python语言的基本语法
- 程序设计基础
- turtle绘图

# THANK YOU

The user can demonstrate on a projector or computer, or print the presentation and make it into a film to be used in a wider field