turtle --- 海龟绘图

源码： Lib/turtle.py

概述

海龟绘图很适合用来引导孩子学习编程。 最初来自于 Wally Feurzeig, Seymour Papert 和 Cynthia Solomon 于 1967 年所创造的 Logo 编程语言。

请想象绘图区有一只机器海龟，起始位置在 x-y 平面的 (0, 0) 点。先执行 `import turtle`，再执行 `turtle.forward(15)`，它将(在屏幕上)朝所面对的 x 轴正方向前进 15 像素，随着它的移动画出一条线段。再执行 `turtle.right(25)`，它将原地右转 25 度。

通过组合使用此类命令，可以轻松地绘制出精美的形状和图案。

`turtle` 模块是基于 Python 标准发行版 2.5 以来的同名模块重新编写并进行了功能扩展。

新模块尽量保持了原模块的特点，并且(几乎)100%与其兼容。这就意味着初学编程者能够以交互方式使用模块的所有命令、类和方法——运行 IDLE 时注意加 `-n` 参数。

turtle 模块提供面向对象和面向过程两种形式的海龟绘图基本组件。由于它使用 `tkinter` 实现基本图形界面，因此需要安装了 Tk 支持的 Python 版本。

面向对象的接口主要使用“2+2”个类：

1. `TurtleScreen` 类定义图形窗口作为绘图海龟的运动场。它的构造器需要一个 `tkinter.Canvas` 或 `ScrolledCanvas` 作为参数。应在 `turtle` 作为某个程序的一部分的时候使用。

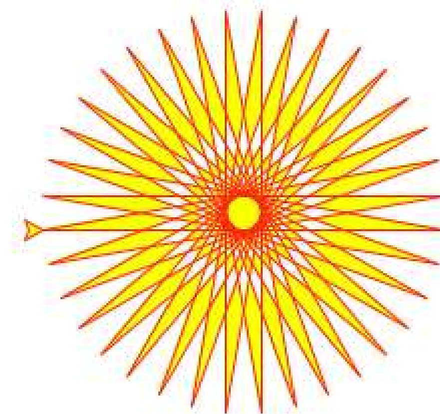
   `Screen()` 函数返回一个 `TurtleScreen` 子类的单例对象。此函数应在 `turtle` 作为独立绘图工具时使用。作为一个单例对象，其所属的类是不可被继承的。

   TurtleScreen/Screen 的所有方法还存在对应的函数，即作为面向过程的接口组成部分。

2. `RawTurtle` (别名：`RawPen`) 类定义海龟对象在 `TurtleScreen` 上绘图。它的构造器需要一个 Canvas, ScrolledCanvas 或 TurtleScreen 作为

Turtle star

使用海龟绘图可以编写重复执行简单动作的程序画出精细复杂的形状。



```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
    forward(200)
    left(170)
    if abs(pos()) < 1:
        break
end_fill()
done()
```

参数，以指定 RawTurtle 对象在哪里绘图。

从 RawTurtle 派生出子类 `Turtle` (别名: `Pen`)，该类对象在 `Screen` 实例上绘图，如果实例不存在则会自动创建。

RawTurtle/Turtle 的所有方法也存在对应的函数，即作为面向过程的接口组成部分。

过程式接口提供与 `Screen` 和 `Turtle` 类的方法相对应的函数。函数名与对应的方法名相同。当 Screen 类的方法对应函数被调用时会自动创建一个 Screen 对象。当 Turtle 类的方法对应函数被调用时会自动创建一个 (匿名的) Turtle 对象。

如果屏幕上需要有多个海龟，就必须使用面向对象的接口。

> **注解:** 以下文档给出了函数的参数列表。对于方法来说当然还有额外的第一个参数 *self*，这里省略了。

可用的 Turtle 和 Screen 方法概览

Turtle 方法

海龟动作

移动和绘制

- `forward()` | `fd()` 前进
- `backward()` | `bk()` | `back()` 后退
- `right()` | `rt()` 右转
- `left()` | `lt()` 左转
- `goto()` | `setpos()` | `setposition()` 前往/定位
- `setx()` 设置x坐标
- `sety()` 设置y坐标
- `setheading()` | `seth()` 设置朝向
- `home()` 返回原点
- `circle()` 画圆
- `dot()` 画点
- `stamp()` 印章
- `clearstamp()` 清除印章
- `clearstamps()` 清除多个印章
- `undo()` 撤消
- `speed()` 速度

获取海龟的状态

- `position()` | `pos()` 位置
- `towards()` 目标方向
- `xcor()` x坐标
- `ycor()` y坐标
- `heading()` 朝向

distance() 距离

设置与度量单位

degrees() 角度
radians() 弧度

画笔控制

绘图状态

pendown() | pd() | down() 画笔落下
penup() | pu() | up() 画笔抬起
pensize() | width() 画笔粗细
pen() 画笔
isdown() 画笔是否落下

颜色控制

color() 颜色
pencolor() 画笔颜色
fillcolor() 填充颜色

填充

filling() 是否填充
begin_fill() 开始填充
end_fill() 结束填充

更多绘图控制

reset() 重置
clear() 清空
write() 书写

海龟状态

可见性

showturtle() | st() 显示海龟
hideturtle() | ht() 隐藏海龟
isvisible() 是否可见

外观

shape() 形状
resizemode() 大小调整模式
shapesize() | turtlesize() 形状大小
shearfactor() 剪切因子
settiltangle() 设置倾角
tiltangle() 倾角
tilt() 倾斜

shapetransform() 变形
get_shapepoly() 获取形状多边形

使用事件

onclick() 当鼠标点击
onrelease() 当鼠标释放
ondrag() 当鼠标拖动

特殊海龟方法

begin_poly() 开始记录多边形
end_poly() 结束记录多边形
get_poly() 获取多边形
clone() 克隆
getturtle() | getpen() 获取海龟画笔
getscreen() 获取屏幕
setundobuffer() 设置撤消缓冲区
undobufferentries() 撤消缓冲区条目数

TurtleScreen/Screen 方法

窗口控制

bgcolor() 背景颜色
bgpic() 背景图片
clear() | clearscreen() 清屏
reset() | resetscreen() 重置
screensize() 屏幕大小
setworldcoordinates() 设置世界坐标系

动画控制

delay() 延迟
tracer() 追踪
update() 更新

使用屏幕事件

listen() 监听
onkey() | onkeyrelease() 当键盘按下并释放
onkeypress() 当键盘按下
onclick() | onscreenclick() 当点击屏幕
ontimer() 当达到定时
mainloop() | done() 主循环

设置与特殊方法

mode() 模式
colormode() 颜色模式

getcanvas() 获取画布
getshapes() 获取形状
register_shape() | addshape() 添加形状
turtles() 所有海龟
window_height() 窗口高度
window_width() 窗口宽度

输入方法

textinput() 文本输入
numinput() 数字输入

Screen 专有方法

bye() 退出
exitonclick() 当点击时退出
setup() 设置
title() 标题

RawTurtle/Turtle 方法和对应函数

本节中的大部分示例都使用 Turtle 类的一个实例，命名为 `turtle`。

海龟动作

turtle.**forward**(*distance*)
turtle.**fd**(*distance*)

参数: **distance** -- 一个数值 (整型或浮点型)

海龟前进 *distance* 指定的距离，方向为海龟的朝向。

```
>>> turtle.position()
(0.00,0.00)
>>> turtle.forward(25)
>>> turtle.position()
(25.00,0.00)
>>> turtle.forward(-75)
>>> turtle.position()
(-50.00,0.00)
```

turtle.**back**(*distance*)
turtle.**bk**(*distance*)
turtle.**backward**(*distance*)

参数: **distance** -- 一个数值

海龟后退 *distance* 指定的距离，方向与海龟的朝向相反。不改变海龟的朝向。

```
>>> turtle.position()
(0.00,0.00)
>>> turtle.backward(30)
>>> turtle.position()
(-30.00,0.00)
```

turtle.**right**(*angle*)
turtle.**rt**(*angle*)

参数: **angle** -- 一个数值 (整型或浮点型)

海龟右转 *angle* 个单位。(单位默认为角度，但可通过 `degrees()` 和 `radians()` 函数改变设置。) 角度的正负由海龟模式确定，参见 `mode()`。

```
>>> turtle.heading()
22.0
>>> turtle.right(45)
>>> turtle.heading()
337.0
```

turtle.**left**(*angle*)
turtle.**lt**(*angle*)

参数: **angle** -- 一个数值 (整型或浮点型)

海龟左转 *angle* 个单位。(单位默认为角度，但可通过 `degrees()` 和 `radians()` 函数改变设置。) 角度的正负由海龟模式确定，参见 `mode()`。

```
>>> turtle.heading()
22.0
>>> turtle.left(45)
>>> turtle.heading()
67.0
```

turtle.**goto**(*x*, *y=None*)
turtle.**setpos**(*x*, *y=None*)
turtle.**setposition**(*x*, *y=None*)

参数:

- **x** -- 一个数值或数值对/向量
- **y** -- 一个数值或 `None`

如果 *y* 为 `None`，*x* 应为一个表示坐标的数值对或 `Vec2D` 类对象 (例如 `pos()` 返回的对象).

海龟移动到一个绝对坐标。如果画笔已落下将会画线。不改变海龟的朝向。

```
>>> tp = turtle.pos()
>>> tp
(0.00,0.00)
>>> turtle.setpos(60,30)
>>> turtle.pos()
(60.00,30.00)
>>> turtle.setpos((20,80))
>>> turtle.pos()
(20.00,80.00)
>>> turtle.setpos(tp)
>>> turtle.pos()
(0.00,0.00)
```

turtle.**setx**(*x*)

参数: **x** -- 一个数值 (整型或浮点型)

设置海龟的横坐标为 *x*，纵坐标保持不变。

```
>>> turtle.position()
(0.00,240.00)
>>> turtle.setx(10)
>>> turtle.position()
(10.00,240.00)
```

turtle.**sety**(*y*)

参数: **y** -- 一个数值 (整型或浮点型)

设置海龟的纵坐标为 *y*，横坐标保持不变。

```
>>> turtle.position()
(0.00,40.00)
>>> turtle.sety(-10)
>>> turtle.position()
(0.00,-10.00)
```

turtle.**setheading**(*to_angle*)
turtle.**seth**(*to_angle*)

参数: **to_angle** -- 一个数值 (整型或浮点型)

设置海龟的朝向为 *to_angle*。以下是以角度表示的几个常用方向:

标准模式	logo 模式
0 - 东	0 - 北
90 - 北	90 - 东

标准模式	logo 模式
180 - 西	180 - 南
270 - 南	270 - 西

```
>>> turtle.setheading(90)
>>> turtle.heading()
90.0
```

turtle.**home**()

海龟移至初始坐标 (0,0)，并设置朝向为初始方向 (由海龟模式确定，参见 `mode()`)。

```
>>> turtle.heading()
90.0
>>> turtle.position()
(0.00,-10.00)
>>> turtle.home()
>>> turtle.position()
(0.00,0.00)
>>> turtle.heading()
0.0
```

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

参数:

- **radius** -- 一个数值
- **extent** -- 一个数值 (或 None)
- **steps** -- 一个整型数 (或 None)

绘制一个 *radius* 指定半径的圆。圆心在海龟左边 *radius* 个单位；*extent* 为一个夹角，用来决定绘制圆的一部分。如未指定 *extent*则绘制整个圆。如果 *extent* 不是完整圆周，则以当前画笔位置为一个端点绘制圆弧。如果 *radius* 为正值则朝逆时针方向绘制圆弧，否则朝顺时针方向。最终海龟的朝向会依据 *extent* 的值而改变。

圆实际是以其内切正多边形来近似表示的，其边的数量由 *steps* 指定。如果未指定边数则会自动确定。此方法也可用来绘制正多边形。

```
>>> turtle.home()
>>> turtle.position()
(0.00,0.00)
>>> turtle.heading()
0.0
>>> turtle.circle(50)
>>> turtle.position()
(-0.00,0.00)
>>> turtle.heading()
0.0
>>> turtle.circle(120, 180)  # draw a semicircle
>>> turtle.position()
```

```
(0.00,240.00)
>>> turtle.heading()
180.0
```

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

参数:
- **size** -- 一个整型数 >= 1 (如果指定)
- **color** -- 一个颜色字符串或颜色数值元组

绘制一个直径为 *size*，颜色为 *color* 的圆点。如果 *size* 未指定，则直径取 pensize+4 和 2*pensize 中的较大值。

```
>>> turtle.home()
>>> turtle.dot()
>>> turtle.fd(50); turtle.dot(20, "blue"); turtle.fd(50)
>>> turtle.position()
(100.00,-0.00)
>>> turtle.heading()
0.0
```

turtle.**stamp**()

在海龟当前位置印制一个海龟形状。返回该印章的 stamp_id，印章可以通过调用 `clearstamp(stamp_id)` 来删除。

```
>>> turtle.color("blue")
>>> turtle.stamp()
11
>>> turtle.fd(50)
```

turtle.**clearstamp**(*stampid*)

参数: **stampid** -- 一个整型数，必须是之前 `stamp()` 调用的返回值

删除 *stampid* 指定的印章。

```
>>> turtle.position()
(150.00,-0.00)
>>> turtle.color("blue")
>>> astamp = turtle.stamp()
>>> turtle.fd(50)
>>> turtle.position()
(200.00,-0.00)
>>> turtle.clearstamp(astamp)
>>> turtle.position()
(200.00,-0.00)
```

turtle.**clearstamps**(*n=None*)

参数: **n** -- 一个整型数 (或 `None`)

删除全部或前/后 *n* 个海龟印章。如果 *n* 为 `None` 则删除全部印章，如果 $n > 0$ 则删除前 *n* 个印章，否则如果 $n < 0$ 则删除后 *n* 个印章。

```
>>> for i in range(8):
...     turtle.stamp(); turtle.fd(30)
13
14
15
16
17
18
19
20
>>> turtle.clearstamps(2)
>>> turtle.clearstamps(-2)
>>> turtle.clearstamps()
```

turtle.**undo()**

撤消 (或连续撤消) 最近的一个 (或多个) 海龟动作。可撤消的次数由撤消缓冲区的大小决定。

```
>>> for i in range(4):
...     turtle.fd(50); turtle.lt(80)
...
>>> for i in range(8):
...     turtle.undo()
```

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

参数: **speed** -- 一个 0..10 范围内的整型数或速度字符串 (见下)

设置海龟移动的速度为 0..10 表示的整型数值。如未指定参数则返回当前速度。

如果输入数值大于 10 或小于 0.5 则速度设为 0。速度字符串与速度值的对应关系如下:

- "fastest": 0 最快
- "fast": 10 快
- "normal": 6 正常
- "slow": 3 慢
- "slowest": 1 最慢

速度值从 1 到 10，画线和海龟转向的动画效果逐级加快。

注意: *speed* = 0 表示 *没有* 动画效果。forward/back 将使海龟向前/向后跳跃，同样的 left/right 将使海龟立即改变朝向。

```
>>> turtle.speed()
3
>>> turtle.speed('normal')
>>> turtle.speed()
6
```

```
>>> turtle.speed(9)
>>> turtle.speed()
9
```

获取海龟的状态

turtle.**position**()
turtle.**pos**()

返回海龟当前的坐标 (x,y) (为 `Vec2D` 矢量类对象)。

```
>>> turtle.pos()
(440.00,-0.00)
```

turtle.**towards**(*x*, *y=None*)

参数:
- **x** -- 一个数值或数值对/矢量，或一个海龟实例
- **y** -- 一个数值——如果 *x* 是一个数值，否则为 `None`

从海龟位置到由 (x,y)，矢量或另一海龟对应位置的连线的夹角。此数值依赖于海龟初始朝向 - 由 "standard"/"world" 或 "logo" 模式设置所决定)。

```
>>> turtle.goto(10, 10)
>>> turtle.towards(0,0)
225.0
```

turtle.**xcor**()

返回海龟的 x 坐标。

```
>>> turtle.home()
>>> turtle.left(50)
>>> turtle.forward(100)
>>> turtle.pos()
(64.28,76.60)
>>> print(round(turtle.xcor(), 5))
64.27876
```

turtle.**ycor**()

返回海龟的 y 坐标。

```
>>> turtle.home()
>>> turtle.left(60)
>>> turtle.forward(100)
>>> print(turtle.pos())
(50.00,86.60)
>>> print(round(turtle.ycor(), 5))
86.60254
```

turtle. **heading**()

返回海龟当前的朝向 (数值依赖于海龟模式参见 `mode()`)。

```
>>> turtle.home()
>>> turtle.left(67)
>>> turtle.heading()
67.0
```

turtle. **distance**(*x*, *y=None*)

参数:

- **x** -- 一个数值或数值对/矢量，或一个海龟实例
- **y** -- 一个数值——如果 *x* 是一个数值，否则为 None

返回从海龟位置到由 (x,y)，适量或另一海龟对应位置的单位距离。

```
>>> turtle.home()
>>> turtle.distance(30,40)
50.0
>>> turtle.distance((30,40))
50.0
>>> joe = Turtle()
>>> joe.forward(77)
>>> turtle.distance(joe)
77.0
```

度量单位设置

turtle. **degrees**(*fullcircle=360.0*)

参数: **fullcircle** -- 一个数值

设置角度的度量单位，即设置一个圆周为多少 "度"。默认值为 360 度。

```
>>> turtle.home()
>>> turtle.left(90)
>>> turtle.heading()
90.0

Change angle measurement unit to grad (also known as gon,
grade, or gradian and equals 1/100-th of the right angle.)
>>> turtle.degrees(400.0)
>>> turtle.heading()
100.0
>>> turtle.degrees(360)
>>> turtle.heading()
90.0
```

turtle. **radians**()

设置角度的度量单位为弧度。其值等于 `degrees(2*math.pi)`。

```
>>> turtle.home()
>>> turtle.left(90)
>>> turtle.heading()
90.0
>>> turtle.radians()
>>> turtle.heading()
1.5707963267948966
```

画笔控制

绘图状态

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

画笔落下 -- 移动时将画线。

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

画笔抬起 -- 移动时不画线。

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

参数: **width** -- 一个正数值

设置线条的粗细为 *width* 或返回该值。如果 resizemode 设为 "auto" 并且 turtleshape 为多边形，该多边形也以同样组细的线条绘制。如未指定参数，则返回当前的 pensize。

```
>>> turtle.pensize()
1
>>> turtle.pensize(10)    # from here on lines of width 10 are drawn
```

turtle.**pen**(*pen=None*, ***pendict*)

参数:
- **pen** -- 一个包含部分或全部下列键的字典
- **pendict** -- 一个或多个以下列键为关键字的关键字参数

返回或设置画笔的属性，以一个包含以下键值对的 "画笔字典" 表示:

- "shown": True/False
- "pendown": True/False
- "pencolor": 颜色字符串或颜色元组

- "fillcolor": 颜色字符串或颜色元组
- "pensize": 正数值
- "speed": 0..10 范围内的数值
- "resizemode": "auto" 或 "user" 或 "noresize"
- "stretchfactor": (正数值, 正数值)
- "outline": 正数值
- "tilt": 数值

此字典可作为后续调用 `pen()` 时的参数，以恢复之前的画笔状态。另外还可将这些属性作为关键词参数提交。使用此方式可以用一条语句设置画笔的多个属性。

```
>>> turtle.pen(fillcolor="black", pencolor="red", pensize=10)
>>> sorted(turtle.pen().items())
[('fillcolor', 'black'), ('outline', 1), ('pencolor', 'red'),
 ('pendown', True), ('pensize', 10), ('resizemode', 'noresize'),
 ('shearfactor', 0.0), ('shown', True), ('speed', 9),
 ('stretchfactor', (1.0, 1.0)), ('tilt', 0.0)]
>>> penstate=turtle.pen()
>>> turtle.color("yellow", "")
>>> turtle.penup()
>>> sorted(turtle.pen().items())[:3]
[('fillcolor', ''), ('outline', 1), ('pencolor', 'yellow')]
>>> turtle.pen(penstate, fillcolor="green")
>>> sorted(turtle.pen().items())[:3]
[('fillcolor', 'green'), ('outline', 1), ('pencolor', 'red')]
```

turtle.**isdown()**

如果画笔落下返回 `True`，如果画笔抬起返回 `False`。

```
>>> turtle.penup()
>>> turtle.isdown()
False
>>> turtle.pendown()
>>> turtle.isdown()
True
```

颜色控制

turtle.**pencolor**(**args*)

返回或设置画笔颜色。

允许以下四种输入格式:

`pencolor()`

返回以颜色描述字符串或元组（见示例）表示的当前画笔颜色。可用作其他 color/pencolor/fillcolor 调用的输入。

`pencolor(colorstring)`

设置画笔颜色为 *colorstring* 指定的 Tk 颜色描述字符串，例如 "red"、"yellow" 或 "#33cc8c"。

pencolor((r, g, b))

设置画笔颜色为以 *r*, *g*, *b* 元组表示的 RGB 颜色。*r*, *g*, *b* 的取值范围应为 0..colormode，colormode 的值为 1.0 或 255 (参见 colormode())。

pencolor(r, g, b)

设置画笔颜色为以 *r*, *g*, *b* 表示的 RGB 颜色。*r*, *g*, *b* 的取值范围应为 0..colormode。

如果 turtleshape 为多边形，该多边形轮廓也以新设置的画笔颜色绘制。

```
>>> colormode()
1.0
>>> turtle.pencolor()
'red'
>>> turtle.pencolor("brown")
>>> turtle.pencolor()
'brown'
>>> tup = (0.2, 0.8, 0.55)
>>> turtle.pencolor(tup)
>>> turtle.pencolor()
(0.2, 0.8, 0.5490196078431373)
>>> colormode(255)
>>> turtle.pencolor()
(51.0, 204.0, 140.0)
>>> turtle.pencolor('#32c18f')
>>> turtle.pencolor()
(50.0, 193.0, 143.0)
```

turtle.**fillcolor**(**args*)

返回或设置填充颜色。

允许以下四种输入格式:

fillcolor()

返回以颜色描述字符串或元组（见示例）表示的当前填充颜色。可用作其他 color/pencolor/fillcolor 调用的输入。

fillcolor(colorstring)

设置填充颜色为 *colorstring* 指定的 Tk 颜色描述字符串，例如 "red"、"yellow" 或 "#33cc8c"。

fillcolor((r, g, b))

设置填充颜色为以 *r*, *g*, *b* 元组表示的 RGB 颜色。*r*, *g*, *b* 的取值范围应为 0..colormode，colormode 的值为 1.0 或 255 (参见 colormode())。

fillcolor(r, g, b)

设置填充颜色为 *r*, *g*, *b* 表示的 RGB 颜色。*r*, *g*, *b* 的取值范围应为 0..colormode。

如果 turtleshape 为多边形，该多边形内部也以新设置的填充颜色填充。

```
>>> turtle.fillcolor("violet")
>>> turtle.fillcolor()
'violet'
>>> turtle.pencolor()
(50.0, 193.0, 143.0)
>>> turtle.fillcolor((50, 193, 143))  # Integers, not floats
>>> turtle.fillcolor()
(50.0, 193.0, 143.0)
>>> turtle.fillcolor('#ffffff')
>>> turtle.fillcolor()
(255.0, 255.0, 255.0)
```

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

返回或设置画笔颜色和填充颜色。

允许多种输入格式。使用如下 0 至 3 个参数:

color()

返回以一对颜色描述字符串或元组表示的当前画笔颜色和填充颜色，两者可分别由 pencolor() 和 fillcolor() 返回。

color(colorstring), color((r,g,b)), color(r,g,b)

输入格式与 pencolor() 相同，同时设置填充颜色和画笔颜色为指定的值。

color(colorstring1, colorstring2), color((r1,g1,b1), (r2,g2,b2))

相当于 pencolor(colorstring1) 加 fillcolor(colorstring2)，使用其他输入格式的方法也与之类似。

如果 turtleshape 为多边形，该多边形轮廓与填充也使用新设置的颜色。

```
>>> turtle.color("red", "green")
>>> turtle.color()
('red', 'green')
>>> color("#285078", "#a0c8f0")
>>> color()
((40.0, 80.0, 120.0), (160.0, 200.0, 240.0))
```

另参见: Screen 方法 colormode()。

填充

turtle.**filling()**

返回填充状态 (填充为 True，否则为 False)。

```
>>> turtle.begin_fill()
>>> if turtle.filling():
...     turtle.pensize(5)
... else:
...     turtle.pensize(3)
```

turtle.**begin_fill()**

在绘制要填充的形状之前调用。

turtle.**end_fill()**

填充上次调用 begin_fill() 之后绘制的形状。

```
>>> turtle.color("black", "red")
>>> turtle.begin_fill()
>>> turtle.circle(80)
>>> turtle.end_fill()
```

更多绘图控制

turtle.**reset()**

从屏幕中删除海龟的绘图，海龟回到原点并设置所有变量为默认值。

```
>>> turtle.goto(0,-22)
>>> turtle.left(100)
>>> turtle.position()
(0.00,-22.00)
>>> turtle.heading()
100.0
>>> turtle.reset()
>>> turtle.position()
(0.00,0.00)
>>> turtle.heading()
0.0
```

turtle.**clear()**

从屏幕中删除指定海龟的绘图。不移动海龟。海龟的状态和位置以及其他海龟的绘图不受影响。

turtle.**write**(*arg*, *move=False*, *align="left"*, *font=("Arial", 8, "normal")*)

参数:

- **arg** -- 要书写到 TurtleScreen 的对象
- **move** -- True/False
- **align** -- 字符串 "left", "center" 或 "right"
- **font** -- 一个三元组 (fontname, fontsize, fonttype)

书写文本 - *arg* 指定的字符串 - 到当前海龟位置，*align* 指定对齐方式 ("left", "center" 或 right")，font 指定字体。如果 *move* 为 True，画笔会移动到文本的右下角。默认 *move* 为 `False`。

```
>>> turtle.write("Home = ", True, align="center")
>>> turtle.write((0,0), True)
```

海龟状态

可见性

turtle.**hideturtle**()
turtle.**ht**()

使海龟不可见。当你绘制复杂图形时这是个好主意，因为隐藏海龟可显著加快绘制速度。

```
>>> turtle.hideturtle()
```

turtle.**showturtle**()
turtle.**st**()

使海龟可见。

```
>>> turtle.showturtle()
```

turtle.**isvisible**()

如果海龟显示返回 `True`，如果海龟隐藏返回 `False`。

```
>>> turtle.hideturtle()
>>> turtle.isvisible()
False
>>> turtle.showturtle()
>>> turtle.isvisible()
True
```

外观

turtle.**shape**(*name=None*)

参数: **name** -- 一个有效的形状名字符串

设置海龟形状为 *name* 指定的形状名，如未指定形状名则返回当前的形状名。*name* 指定的形状名应存在于 TurtleScreen 的 shape 字典中。多边形的形状初始时有以下几种："arrow", "turtle", "circle", "square", "triangle", "classic"。要了解如何处理形状请参看 Screen 方法 `register_shape()`。

```
>>> turtle.shape()
'classic'
>>> turtle.shape("turtle")
>>> turtle.shape()
'turtle'
```

turtle.**resizemode**(*rmode=None*)

参数: **rmode** -- 字符串 "auto", "user", "noresize" 其中之一

设置大小调整模式为以下值之一: "auto", "user", "noresize"。如未指定 *rmode* 则返回当前的大小调整模式。不同的大小调整模式的效果如下:

- "auto": 根据画笔粗细值调整海龟的外观。
- "user": 根据拉伸因子和轮廓宽度 (outline) 值调整海龟的外观，两者是由 shapesize() 设置的。
- "noresize": 不调整海龟的外观大小。

大小调整模式 ("user") 会在 shapesize() 带参数调用时生效。

```
>>> turtle.resizemode()
'noresize'
>>> turtle.resizemode("auto")
>>> turtle.resizemode()
'auto'
```

turtle.**shapesize**(*stretch_wid=None*, *stretch_len=None*, *outline=None*)
turtle.**turtlesize**(*stretch_wid=None*, *stretch_len=None*, *outline=None*)

参数:
- **stretch_wid** -- 正数值
- **stretch_len** -- 正数值
- **outline** -- 正数值

返回或设置画笔的属性 x/y-拉伸因子和/或轮廓。设置大小调整模式为 "user"。当且仅当大小调整模式设为 "user" 时海龟会基于其拉伸因子调整外观: *stretch_wid* 为垂直于其朝向的宽度拉伸因子，*stretch_len* 为平等于其朝向的长度拉伸因子，决定形状轮廓线的粗细。

```
>>> turtle.shapesize()
(1.0, 1.0, 1)
>>> turtle.resizemode("user")
>>> turtle.shapesize(5, 5, 12)
>>> turtle.shapesize()
(5, 5, 12)
>>> turtle.shapesize(outline=8)
>>> turtle.shapesize()
(5, 5, 8)
```

turtle.**shearfactor**(*shear=None*)

参数: **shear** -- 数值 (可选)

设置或返回当前的剪切因子。根据 share 指定的剪切因子即剪切角度的切线来剪切海龟形状。*不*改变海龟的朝向 (移动方向)。如未指定 shear 参数: 返回当前的剪切因子即剪切角度的切线，与海龟朝向平行的线条将被剪切。

```
>>> turtle.shape("circle")
>>> turtle.shapesize(5,2)
>>> turtle.shearfactor(0.5)
>>> turtle.shearfactor()
0.5
```

turtle.**tilt**(*angle*)

参数: **angle** -- 一个数值

海龟形状自其当前的倾角转动 *angle* 指定的角度，但 *不*改变海龟的朝向 (移动方向)。

```
>>> turtle.reset()
>>> turtle.shape("circle")
>>> turtle.shapesize(5,2)
>>> turtle.tilt(30)
>>> turtle.fd(50)
>>> turtle.tilt(30)
>>> turtle.fd(50)
```

turtle.**settiltangle**(*angle*)

参数: **angle** -- 一个数值

旋转海龟形状使其指向 *angle* 指定的方向，忽略其当前的倾角，*不*改变海龟的朝向 (移动方向)。

```
>>> turtle.reset()
>>> turtle.shape("circle")
>>> turtle.shapesize(5,2)
>>> turtle.settiltangle(45)
>>> turtle.fd(50)
>>> turtle.settiltangle(-45)
>>> turtle.fd(50)
```

3.1 版后已移除.

turtle.**tiltangle**(*angle=None*)

参数: **angle** -- 一个数值 (可选)

设置或返回当前的倾角。如果指定 angle 则旋转海龟形状使其指向 angle 指定的方向，忽略其当前的倾角。*不*改变海龟的朝向 (移动方向)。如果未指定 angle: 返回当前的倾角，即海龟形

状的方向和海龟朝向 (移动方向) 之间的夹角。

```
>>> turtle.reset()
>>> turtle.shape("circle")
>>> turtle.shapesize(5,2)
>>> turtle.tilt(45)
>>> turtle.tiltangle()
45.0
```

turtle.**shapetransform**(*t11=None*, *t12=None*, *t21=None*, *t22=None*)

参数:

- **t11** -- 一个数值 (可选)
- **t12** -- 一个数值 (可选)
- **t21** -- 一个数值 (可选)
- **t12** -- 一个数值 (可选)

设置或返回海龟形状的当前变形矩阵。

如不指定任何矩阵元素，则返回以4元素元组表示的变形矩阵。否则使用指定元素设置变形矩阵改变海龟形状，矩阵第一排的值为 t11, t12，第二排的值为 t21, t22。行列式 t11 * t22 - t12 * t21 的值不能为零，否则会出错。根据指定的矩阵修改拉伸因子，剪切因子和倾角。

```
>>> turtle = Turtle()
>>> turtle.shape("square")
>>> turtle.shapesize(4,2)
>>> turtle.shearfactor(-0.5)
>>> turtle.shapetransform()
(4.0, -1.0, -0.0, 2.0)
```

turtle.**get_shapepoly**()

返回以坐标值对元组表示的当前形状多边形。这可以用于定义一个新形状或一个复合形状的多个组成部分。

```
>>> turtle.shape("square")
>>> turtle.shapetransform(4, -1, 0, 2)
>>> turtle.get_shapepoly()
((50, -20), (30, 20), (-50, 20), (-30, -20))
```

使用事件

turtle.**onclick**(*fun*, *btn=1*, *add=None*)

参数:

- **fun** -- 一个函数，调用时将传入两个参数表示在画布上点击的坐标。
- **btn** -- 鼠标按钮编号，默认值为 1 (鼠标左键)
- **add** -- `True` 或 `False` -- 如为 `True` 则将添加一个新绑定，否则将取代先前的绑定

将 *fun* 指定的函数绑定到鼠标点击此海龟事件。如果 *fun* 值为 `None`，则移除现有的绑定。以下为使用匿名海龟即过程式的示例:

```
>>> def turn(x, y):
...     left(180)
...
>>> onclick(turn)  # Now clicking into the turtle will turn it.
>>> onclick(None)  # event-binding will be removed
```

turtle.**onrelease**(*fun*, *btn=1*, *add=None*)

参数:

- **fun** -- 一个函数，调用时将传入两个参数表示在画布上点击的坐标。
- **btn** -- 鼠标按钮编号，默认值为 1 (鼠标左键)
- **add** -- `True` 或 `False` -- 如为 `True` 则将添加一个新绑定，否则将取代先前的绑定

将 *fun* 指定的函数绑定到在此海龟上释放鼠标按键事件。如果 *fun* 值为 `None`，则移除现有的绑定。

```
>>> class MyTurtle(Turtle):
...     def glow(self,x,y):
...         self.fillcolor("red")
...     def unglow(self,x,y):
...         self.fillcolor("")
...
>>> turtle = MyTurtle()
>>> turtle.onclick(turtle.glow)     # clicking on turtle turns fillcolor red,
>>> turtle.onrelease(turtle.unglow) # releasing turns it to transparent.
```

turtle.**ondrag**(*fun*, *btn=1*, *add=None*)

参数:

- **fun** -- 一个函数，调用时将传入两个参数表示在画布上点击的坐标。
- **btn** -- 鼠标按钮编号，默认值为 1 (鼠标左键)
- **add** -- `True` 或 `False` -- 如为 `True` 则将添加一个新绑定，否则将取代先前的绑定

将 *fun* 指定的函数绑定到在此海龟上移动鼠标事件。如果 *fun* 值为 `None`，则移除现有的绑定。

注: 在海龟上移动鼠标事件之前应先发生在此海龟上点击鼠标事件。

```
>>> turtle.ondrag(turtle.goto)
```

在此之后点击并拖动海龟可在屏幕上手绘线条 (如果画笔为落下)。

特殊海龟方法

turtle. **begin_poly()**

开始记录多边形的顶点。当前海龟位置为多边形的第一个顶点。

turtle. **end_poly()**

停止记录多边形的顶点。当前海龟位置为多边形的最后一个顶点。它将连线到第一个顶点。

turtle. **get_poly()**

返回最新记录的多边形。

```
>>> turtle.home()
>>> turtle.begin_poly()
>>> turtle.fd(100)
>>> turtle.left(20)
>>> turtle.fd(30)
>>> turtle.left(60)
>>> turtle.fd(50)
>>> turtle.end_poly()
>>> p = turtle.get_poly()
>>> register_shape("myFavouriteShape", p)
```

turtle. **clone()**

创建并返回海龟的克隆体，具有相同的位置、朝向和海龟属性。

```
>>> mick = Turtle()
>>> joe = mick.clone()
```

turtle. **getturtle()**
turtle. **getpen()**

返回海龟对象自身。唯一合理的用法: 作为一个函数来返回 "匿名海龟":

```
>>> pet = getturtle()
>>> pet.fd(50)
>>> pet
<turtle.Turtle object at 0x...>
```

turtle. **getscreen()**

返回作为海龟绘图场所的 `TurtleScreen` 类对象。该对象将可调用 TurtleScreen 方法。

```
>>> ts = turtle.getscreen()
>>> ts
<turtle._Screen object at 0x...>
>>> ts.bgcolor("pink")
```

turtle. **setundobuffer**(*size*)

参数: **size** -- 一个整型数值或 None

设置或禁用撤消缓冲区。如果 *size* 为一个整型数则将开辟一个指定大小的空缓冲区。*size* 表示可使用 undo() 方法/函数撤消的海龟命令的次数上限。如果 *size* 为 None 则禁用撤消缓冲区。

```
>>> turtle.setundobuffer(42)
```

turtle.**undobufferentries()**

返回撤销缓冲区里的条目数。

```
>>> while undobufferentries():
...     undo()
```

复合形状

要使用由多个不同颜色多边形构成的复合海龟形状，你必须明确地使用辅助类 Shape，具体步骤如下:

1. 创建一个空 Shape 对象，类型为 "compound"。
2. 按照需要使用 addcomponent() 方法向此对象添加多个部件。

   例如:

```
>>> s = Shape("compound")
>>> poly1 = ((0,0),(10,-5),(0,10),(-10,-5))
>>> s.addcomponent(poly1, "red", "blue")
>>> poly2 = ((0,0),(10,-5),(-10,-5))
>>> s.addcomponent(poly2, "blue", "red")
```

3. 接下来将 Shape 对象添加到 Screen 对象的形状列表并使用它:

```
>>> register_shape("myshape", s)
>>> shape("myshape")
```

注解: Shape 类在 register_shape() 方法的内部以多种方式使用。应用程序编写者 *只有* 在使用上述的复合形状时才需要处理 Shape 类。

TurtleScreen/Screen 方法及对应函数

本节中的大部分示例都使用 TurtleScreen 类的一个实例，命名为 screen。

窗口控制

turtle.**bgcolor**(**args*)

参数: **args** -- 一个颜色字符串或三个取值范围 0..colormode 内的数值或一个取值范围相同的数值3元组

设置或返回 TurtleScreen 的背景颜色。

```
>>> screen.bgcolor("orange")
>>> screen.bgcolor()
'orange'
>>> screen.bgcolor("#800080")
>>> screen.bgcolor()
(128.0, 0.0, 128.0)
```

turtle.**bgpic**(*picname=None*)

参数: **picname** -- 一个字符串, gif-文件名, `"nopic"`, 或 `None`

设置背景图片或返回当前背景图片名称。如果 *picname* 为一个文件名，则将相应图片设为背景。如果 *picname* 为 `"nopic"`，则删除当前背景图片。如果 *picname* 为 `None`，则返回当前背景图片文件名。：

```
>>> screen.bgpic()
'nopic'
>>> screen.bgpic("landscape.gif")
>>> screen.bgpic()
"landscape.gif"
```

turtle.**clear**()
turtle.**clearscreen**()

从中删除所有海龟的全部绘图。将已清空的 TurtleScreen 重置为初始状态: 白色背景，无背景片，无事件绑定并启用追踪。

> **注解：** 此 TurtleScreen 方法作为全局函数时只有一个名字 `clearscreen`。全局函数 `clear` 所对应的是 Turtle 方法 `clear`。

turtle.**reset**()
turtle.**resetscreen**()

重置屏幕上的所有海龟为其初始状态。

> **注解：** 此 TurtleScreen 方法作为全局函数时只有一个名字 `resetscreen`。全局函数 `reset` 所对应的是 Turtle 方法 `reset`。

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

参数:
- **canvwidth** -- 正整型数，以像素表示画布的新宽度值

- **canvheight** -- 正整型数，以像素表示画面的新高度值
- **bg** -- 颜色字符串或颜色元组，新的背景颜色

如未指定任何参数，则返回当前的 (canvaswidth, canvasheight)。否则改变作为海龟绘图场所的画布大小。不改变绘图窗口。要观察画布的隐藏区域，可以使用滚动条。通过此方法可以令之前绘制于画布之外的图形变为可见。

```
>>> screen.screensize()
(400, 300)
>>> screen.screensize(2000,1500)
>>> screen.screensize()
(2000, 1500)
```

也可以用来寻找意外逃走的海龟 ;-)

turtle.**setworldcoordinates**(*llx*, *lly*, *urx*, *ury*)

参数:
- **llx** -- 一个数值, 画布左下角的 x-坐标
- **lly** -- 一个数值, 画布左下角的 y-坐标
- **urx** -- 一个数值, 画面右上角的 x-坐标
- **ury** -- 一个数值, 画布右上角的 y-坐标

设置用户自定义坐标系并在必要时切换模式为 "world"。这会执行一次 `screen.reset()`。如果 "world" 模式已激活，则所有图形将根据新的坐标系重绘。

注意: 在用户自定义坐标系中，角度可能显得扭曲。

```
>>> screen.reset()
>>> screen.setworldcoordinates(-50,-7.5,50,7.5)
>>> for _ in range(72):
...     left(10)
...
>>> for _ in range(8):
...     left(45); fd(2)   # a regular octagon
```

动画控制

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

参数: **delay** -- 正整型数

设置或返回以毫秒数表示的延迟值 *delay*。(这约等于连续两次画布刷新的间隔时间。) 绘图延迟越长，动画速度越慢。

可选参数:

```
>>> screen.delay()
10
```

```
>>> screen.delay(5)
>>> screen.delay()
5
```

turtle.**tracer**(*n=None*, *delay=None*)

参数:

- **n** -- 非负整型数
- **delay** -- 非负整型数

启用/禁用海龟动画并设置刷新图形的延迟时间。如果指定 *n* 值，则只有每第 n 次屏幕刷新会实际执行。(可被用来加速复杂图形的绘制。) 如果调用时不带参数，则返回当前保存的 n 值。第二个参数设置延迟值 (参见 `delay()`)。

```
>>> screen.tracer(8, 25)
>>> dist = 2
>>> for i in range(200):
...     fd(dist)
...     rt(90)
...     dist += 2
```

turtle.**update**()

执行一次 TurtleScreen 刷新。在禁用追踪时使用。

另参见 RawTurtle/Turtle 方法 `speed()`。

使用屏幕事件

turtle.**listen**(*xdummy=None*, *ydummy=None*)

设置焦点到 TurtleScreen (以便接收按键事件)。使用两个 Dummy 参数以便能够传递 `listen()` 给 onclick 方法。

turtle.**onkey**(*fun*, *key*)
turtle.**onkeyrelease**(*fun*, *key*)

参数:

- **fun** -- 一个无参数的函数或 `None`
- **key** -- 一个字符串: 键 (例如 "a") 或键标 (例如 "space")

绑定 *fun* 指定的函数到按键释放事件。如果 *fun* 值为 `None`，则移除事件绑定。注: 为了能够注册按键事件，TurtleScreen 必须得到焦点。(参见 method `listen()` 方法。)

```
>>> def f():
...     fd(50)
...     lt(60)
...
>>> screen.onkey(f, "Up")
>>> screen.listen()
```

turtle.**onkeypress**(*fun*, *key=None*)

参数:
- **fun** -- 一个无参数的函数或 None
- **key** -- 一个字符串: 键 (例如 "a") 或键标 (例如 "space")

绑定 *fun* 指定的函数到指定键的按下事件。如未指定键则绑定到任意键的按下事件。注: 为了能够注册按键事件，必须得到焦点。(参见 listen() 方法。)

```
>>> def f():
...     fd(50)
...
>>> screen.onkey(f, "Up")
>>> screen.listen()
```

turtle.**onclick**(*fun*, *btn=1*, *add=None*)
turtle.**onscreenclick**(*fun*, *btn=1*, *add=None*)

参数:
- **fun** -- 一个函数，调用时将传入两个参数表示在画布上点击的坐标。
- **btn** -- 鼠标按钮编号，默认值为 1 (鼠标左键)
- **add** -- True 或 False -- 如为 True 则将添加一个新绑定，否则将取代先前的绑定

绑定 *fun* 指定的函数到鼠标点击屏幕事件。如果 *fun* 值为 None，则移除现有的绑定。

以下示例使用一个 TurtleScreen 实例 screen 和一个 Turtle 实例 turtle:

```
>>> screen.onclick(turtle.goto) # Subsequently clicking into the TurtleScreen will
>>>                             # make the turtle move to the clicked point.
>>> screen.onclick(None)        # remove event binding again
```

注解: 此 TurtleScreen 方法作为全局函数时只有一个名字 onscreenclick。全局函数 onclick 所对应的是 Turtle 方法 onclick。

turtle.**ontimer**(*fun*, *t=0*)

参数:
- **fun** -- 一个无参数的函数
- **t** -- 一个数值 >= 0

安装一个计时器，在 *t* 毫秒后调用 *fun* 函数。

```
>>> running = True
>>> def f():
...     if running:
...         fd(50)
...         lt(60)
...         screen.ontimer(f, 250)
```

```
>>> f()   ### makes the turtle march around
>>> running = False
```

turtle.**mainloop**()
turtle.**done**()

开始事件循环 - 调用 Tkinter 的 mainloop 函数。必须作为一个海龟绘图程序的结束语句。如果一个脚本是在以 -n 模式 (无子进程) 启动的 IDLE 中运行时 *不可* 使用 - 用于实现海龟绘图的交互功能。:

```
>>> screen.mainloop()
```

输入方法

turtle.**textinput**(*title*, *prompt*)

参数:
- **title** -- 字符串
- **prompt** -- 字符串

弹出一个对话框窗口用来输入一个字符串。形参 title 为对话框窗口的标题，prompt 为一条文本，通常用来提示要输入什么信息。返回输入的字符串。如果对话框被取消则返回 `None`。:

```
>>> screen.textinput("NIM", "Name of first player:")
```

turtle.**numinput**(*title*, *prompt*, *default=None*, *minval=None*, *maxval=None*)

参数:
- **title** -- 字符串
- **prompt** -- 字符串
- **default** -- 数值 (可选)
- **minval** -- 数值 (可选)
- **maxval** -- 数值 (可选)

弹出一个对话框窗口用来输入一个数值。title 为对话框窗口的标题，prompt 为一条文本，通常用来描述要输入的数值信息。default: 默认值, minval: 可输入的最小值, maxval: 可输入的最大值。输入数值的必须在指定的 minval .. maxval 范围之内，否则将给出一条提示，对话框保持打开等待修改。返回输入的数值。如果对话框被取消则返回 `None`。:

```
>>> screen.numinput("Poker", "Your stakes:", 1000, minval=10, maxval=10000)
```

设置与特殊方法

turtle.**mode**(*mode=None*)

参数: **mode** -- 字符串 "standard", "logo" 或 "world" 其中之一

设置海龟模式 ("standard", "logo" 或 "world") 并执行重置。如未指定模式则返回当前的模式。

"standard" 模式与旧的 `turtle` 兼容。"logo" 模式与大部分 Logo 海龟绘图兼容。"world" 模式使用用户自定义的 "世界坐标系"。**注意**: 在此模式下，如果 `x/y` 单位比率不等于 1 则角度会显得扭曲。

模式	初始海龟朝向	正数角度
"standard"	朝右 (东)	逆时针
"logo"	朝上 (北)	顺时针

```
>>> mode("logo")    # resets turtle heading to north
>>> mode()
'logo'
```

turtle.**colormode**(*cmode=None*)

参数: **cmode** -- 数值 1.0 或 255 其中之一

返回颜色模式或将其设为 1.0 或 255。构成颜色三元组的 *r*, *g*, *b* 数值必须在 0..*cmode* 范围之内。

```
>>> screen.colormode(1)
>>> turtle.pencolor(240, 160, 80)
Traceback (most recent call last):
     ...
TurtleGraphicsError: bad color sequence: (240, 160, 80)
>>> screen.colormode()
1.0
>>> screen.colormode(255)
>>> screen.colormode()
255
>>> turtle.pencolor(240,160,80)
```

turtle.**getcanvas**()

返回此 TurtleScreen 的 Canvas 对象。供了解 Tkinter 的 Canvas 对象内部机理的人士使用。

```
>>> cv = screen.getcanvas()
>>> cv
<turtle.ScrolledCanvas object ...>
```

turtle.**getshapes**()

返回所有当前可用海龟形状的列表。

```
>>> screen.getshapes()
['arrow', 'blank', 'circle', ..., 'turtle']
```

turtle.**register_shape**(*name*, *shape=None*)

turtle.**addshape**(*name*, *shape=None*)

调用此函数有三种不同方式:

1. *name* 为一个 gif 文件的文件名， *shape* 为 None: 安装相应的图像形状。:

```
>>> screen.register_shape("turtle.gif")
```

注解: 当海龟转向时图像形状 *不会* 转动，因此无法显示海龟的朝向!

2. *name* 为指定的字符串，*shape* 为由坐标值对构成的元组: 安装相应的多边形形状。

```
>>> screen.register_shape("triangle", ((5,-3), (0,5), (-5,-3)))
```

3. *name* 为指定的字符串， 为一个 (复合) Shape 类对象: 安装相应的复合形状。

将一个海龟形状加入 TurtleScreen 的形状列表。只有这样注册过的形状才能通过执行 shape(shapename) 命令来使用。

turtle.**turtles**()

返回屏幕上的海龟列表。

```
>>> for turtle in screen.turtles():
...     turtle.color("red")
```

turtle.**window_height**()

返回海龟窗口的高度。:

```
>>> screen.window_height()
480
```

turtle.**window_width**()

返回海龟窗口的宽度。:

```
>>> screen.window_width()
640
```

Screen 专有方法, 而非继承自 TurtleScreen

turtle.**bye**()

关闭海龟绘图窗口。

turtle.**exitonclick**()

将 bye() 方法绑定到 Screen 上的鼠标点击事件。

如果配置字典中 "using_IDLE" 的值为 `False` (默认值) 则同时进入主事件循环。注: 如果启动 IDLE 时使用了 `-n` 开关 (无子进程)，`turtle.cfg` 中此数值应设为 `True`。在此情况下 IDLE 本身的主事件循环同样会作用于客户脚本。

turtle.**setup**(*width=_CFG["width"], height=_CFG["height"], startx=_CFG["leftright"], starty=_CFG["topbottom"]*)

设置主窗口的大小和位置。默认参数值保存在配置字典中，可通过 `turtle.cfg` 文件进行修改。

参数:

- **width** -- 如为一个整型数值，表示大小为多少像素，如为一个浮点数值，则表示屏幕的占比；默认为屏幕的 50%
- **height** -- 如为一个整型数值，表示高度为多少像素，如为一个浮点数值，则表示屏幕的占比；默认为屏幕的 75%
- **startx** -- 如为正值，表示初始位置距离屏幕左边缘多少像素，负值表示距离右边缘，`None` 表示窗口水平居中
- **starty** -- 如为正值，表示初始位置距离屏幕上边缘多少像素，负值表示距离下边缘，`None` 表示窗口垂直居中

```
>>> screen.setup (width=200, height=200, startx=0, starty=0)
>>>              # sets window to 200x200 pixels, in upper left of screen
>>> screen.setup(width=.75, height=0.5, startx=None, starty=None)
>>>              # sets window to 75% of screen by 50% of screen and centers
```

turtle.**title**(*titlestring*)

参数: **titlestring** -- 一个字符串，显示为海龟绘图窗口的标题栏文本

设置海龟窗口标题为 *titlestring* 指定的文本。

```
>>> screen.title("Welcome to the turtle zoo!")
```

公共类

class turtle.**RawTurtle**(*canvas*)
class turtle.**RawPen**(*canvas*)

参数: **canvas** -- 一个 `tkinter.Canvas` , `ScrolledCanvas` 或 `TurtleScreen` 类对象

创建一个海龟。海龟对象具有 "Turtle/RawTurtle 方法" 一节所述的全部方法。

class turtle.**Turtle**

RawTurtle 的子类，具有相同的接口，但其绘图场所为默认的 `Screen` 类对象，在首次使用时自动创建。

class turtle.**TurtleScreen**(*cv*)

参数: **cv** -- 一个 tkinter.Canvas 类对象

提供面向屏幕的方法例如 setbg() 等。说明见上文。

class turtle.**Screen**

TurtleScreen 的子类，增加了四个方法.

class turtle.**ScrolledCanvas**(*master*)

参数: **master** -- 可容纳 ScrolledCanvas 的 Tkinter 部件，即添加了滚动条的 Tkinter-canvas

由 Screen 类使用，使其能够自动提供一个 ScrolledCanvas 作为海龟的绘图场所。

class turtle.**Shape**(*type_*, *data*)

参数: **type_** -- 字符串 "polygon", "image", "compound" 其中之一

实现形状的数据结构。(type_, data) 必须遵循以下定义:

*type_*	*data*
"polygon"	一个多边形元组，即由坐标值对构成的元组
"image"	一个图片 (此形式仅限内部使用!)
"compound"	None (复合形状必须使用 addcomponent() 方法来构建)

addcomponent(*poly*, *fill*, *outline=None*)

参数:
- **poly** -- 一个多边形，即由数值对构成的元组
- **fill** -- 一种颜色，将用来填充 *poly* 指定的多边形
- **outline** -- 一种颜色，用于多边形的轮廓 (如有指定)

示例:

```
>>> poly = ((0,0),(10,-5),(0,10),(-10,-5))
>>> s = Shape("compound")
>>> s.addcomponent(poly, "red", "blue")
>>> # ... add more components and then use register_shape()
```

参见 复合形状。

class turtle.**Vec2D**(*x*, *y*)

一个二维矢量类，用来作为实现海龟绘图的辅助类。也可能在海龟绘图程序中使用。派生自元组，因此矢量也属于元组!

提供的运算 (*a*, *b* 为矢量, *k* 为数值):

- `a + b` 矢量加法
- `a - b` 矢量减法
- `a * b` 内积
- `k * a` 和 `a * k` 与标量相乘
- `abs(a)` a 的绝对值
- `a.rotate(angle)` 旋转

帮助与配置

如何使用帮助

Screen 和 Turtle 类的公用方法以文档字符串提供了详细的文档。因此可以利用 Python 帮助工具获取这些在线帮助信息:

- 当使用 IDLE 时，输入函数/方法调用将弹出工具提示显示其签名和文档字符串的头几行。
- 对文法或函数调用 `help()` 将显示其文档字符串:

```
>>> help(Screen.bgcolor)
Help on method bgcolor in module turtle:

bgcolor(self, *args) unbound turtle.Screen method
    Set or return backgroundcolor of the TurtleScreen.

    Arguments (if given): a color string or three numbers
    in the range 0..colormode or a 3-tuple of such numbers.


      >>> screen.bgcolor("orange")
      >>> screen.bgcolor()
      "orange"
      >>> screen.bgcolor(0.5,0,0.5)
      >>> screen.bgcolor()
      "#800080"

>>> help(Turtle.penup)
Help on method penup in module turtle:

penup(self) unbound turtle.Turtle method
    Pull the pen up -- no drawing when moving.

    Aliases: penup | pu | up

    No argument

    >>> turtle.penup()
```

- 方法对应函数的文档字符串的形式会有一些修改:

```
>>> help(bgcolor)
Help on function bgcolor in module turtle:

bgcolor(*args)
    Set or return backgroundcolor of the TurtleScreen.

    Arguments (if given): a color string or three numbers
    in the range 0..colormode or a 3-tuple of such numbers.

    Example::

      >>> bgcolor("orange")
      >>> bgcolor()
      "orange"
      >>> bgcolor(0.5,0,0.5)
      >>> bgcolor()
      "#800080"

>>> help(penup)
Help on function penup in module turtle:

penup()
    Pull the pen up -- no drawing when moving.

    Aliases: penup | pu | up

    No argument

    Example:
    >>> penup()
```

这些修改版文档字符串是在导入时与方法对应函数的定义一起自动生成的。

文档字符串翻译为不同的语言

可使用工具创建一个字典，键为方法名，值为 Screen 和 Turtle 类公共方法的文档字符串。

turtle.**write_docstringdict**(*filename="turtle_docstringdict"*)

参数: **filename** -- 一个字符串，表示文件名

创建文档字符串字典并将其写入 filename 指定的 Python 脚本文件。此函数必须显示地调用 (海龟绘图类并不使用此函数)。文档字符串字典将被写入到 Python 脚本文件 `filename.py`。该文件可作为模板用来将文档字符串翻译为不同语言。

如果你 (或你的学生) 想使用本国语言版本的 `turtle` 在线帮助，你必须翻译文档字符串并保存结果文件，例如 `turtle_docstringdict_german.py`.

如果你在 `turtle.cfg` 文件中加入了相应的条目，此字典将在导入模块时被读取并替代原有的英文版文档字符串。

在撰写本文档时已经有了德语和意大利语版的文档字符串字典。(更多需求请联系 glingl@aon.at)

如何配置 Screen 和 Turtle

内置的默认配置是模仿旧 turtle 模块的外观和行为，以便尽可能地与其保持兼容。

如果你想使用不同的配置，以便更好地反映此模块的特性或是更适合你的需求，例如在课堂中使用，你可以准备一个配置文件 `turtle.cfg`，该文件将在导入模块时被读取并根据其中的设定修改模块配置。

内置的配置对应以下的 turtle.cfg:

```
width = 0.5
height = 0.75
leftright = None
topbottom = None
canvwidth = 400
canvheight = 300
mode = standard
colormode = 1.0
delay = 10
undobuffersize = 1000
shape = classic
pencolor = black
fillcolor = black
resizemode = noresize
visible = True
language = english
exampleturtle = turtle
examplescreen = screen
title = Python Turtle Graphics
using_IDLE = False
```

选定条目的简短说明:

- 开头的四行对应 `Screen.setup()` 方法的参数。
- 第 5 和 6 行对应 `Screen.screensize()` 方法的参数。
- *shape* 可以是任何内置形状，即: arrow, turtle 等。更多信息可用 `help(shape)` 查看。
- 如果你想使用无填充色 (即令海龟变透明)，你必须写 `fillcolor = ""` (但 cfg 文件中所有非空字符串都不可加引号)。
- 如果你想令海龟反映其状态，你必须使用 `resizemode = auto`。
- 如果你设置语言例如 `language = italian` 则文档字符串字典 `turtle_docstringdict_italian.py` 将在导入模块时被加载 (如果导入路径即 `turtle` 的目录中存在此文件。
- *exampleturtle* 和 *examplescreen* 条目定义了相应对象在文档字符串中显示的名称。方法文档字符串转换为函数文档字符串时将从文档字符串中删去这些名称。
- *using_IDLE*: 如果你经常使用 IDLE 并启用其 -n 开关 ("无子进程") 则应将此项设为 `True`，这将阻止 `exitonclick()` 进入主事件循环。

turtle.cfg 文件可以保存于 turtle 所在目录，当前工作目录也可以有一个同名文件。后者会重载覆盖前者的设置。

Lib/turtledemo 目录中也有一个 turtle.cfg 文件。你可以将其作为示例进行研究，并在运行演示时查看其作用效果 (但最好不要在演示查看器中运行)。

turtledemo --- 演示脚本集

turtledemo 包汇集了一组演示脚本。这些脚本可以通过以下命令打开所提供的演示查看器运行和查看:

```
python -m turtledemo
```

此外，你也可以单独运行其中的演示脚本。例如，:

```
python -m turtledemo.bytedesign
```

turtledemo 包目录中的内容:

- 一个演示查看器 __main__.py，可用来查看脚本的源码并即时运行。
- 多个脚本文件，演示 turtle 模块的不同特性。所有示例可通过 Examples 菜单打开。也可以单独运行每个脚本。
- 一个 turtle.cfg 文件，作为说明如何编写并使用模块配置文件的示例模板。

演示脚本清单如下:

名称	描述	相关特性
bytedesign	复杂的传统海龟绘图模式	tracer(), delay, update()
chaos	绘制 Verhulst 动态模型，演示通过计算机的运算可能会生成令人惊叹的结果	世界坐标系
clock	绘制模拟时钟显示本机的当前时间	海龟作为表针, ontimer
colormixer	试验 r, g, b 颜色模式	ondrag() 当鼠标拖动
forest	绘制 3 棵广度优先树	随机化
fractalcurves	绘制 Hilbert & Koch 曲线	递归
lindenmayer	文化数学 (印度装饰艺术)	L-系统
minimal_hanoi	汉诺塔	矩形海龟作为汉诺盘 (shape, shapesize)
nim	玩经典的“尼姆”游戏，开始时有三堆小棒，与电脑对战。	海龟作为小棒，事件驱动 (鼠标, 键盘)
paint	超极简主义绘画程序	onclick() 当鼠标点击

名称	描述	相关特性
peace	初级技巧	海龟: 外观与动画
penrose	非周期性地使用风筝和飞镖形状铺满平面	`stamp()` 印章
planet_and_moon	模拟引力系统	复合开关, `Vec2D` 类
round_dance	两两相对并不断旋转舞蹈的海龟	复合形状, clone shapesize, tilt, get_shapepoly, update
sorting_animate	动态演示不同的排序方法	简单对齐, 随机化
tree	一棵 (图形化的) 广度优先树 (使用生成器)	`clone()` 克隆
two_canvases	简单设计	两块画布上的海龟
wikipedia	一个来自介绍海龟绘图的维基百科文章的图案	`clone()`, `undo()`
yinyang	另一个初级示例	`circle()` 画圆

祝你玩得开心!

Python 2.6 之后的变化

- `Turtle.tracer()`, `Turtle.window_width()` 和 `Turtle.window_height()` 方法已被去除。具有这些名称和功能的方法现在只限于 `Screen` 类的方法。但其对应的函数仍然可用。(实际上在 Python 2.6 中这些方法就已经只是从对应的 `TurtleScreen`/`Screen` 类的方法复制而来。)
- `Turtle.fill()` 方法已被去除。`begin_fill()` 和 `end_fill()` 的行为则有细微改变: 现在每个填充过程必须以一个 `end_fill()` 调用来结束。
- 新增了一个 `Turtle.filling()` 方法。该方法返回一个布尔值: 如果填充过程正在进行为 `True`，否则为 `False`。此行为相当于 Python 2.6 中不带参数的 `fill()` 调用。

Python 3.0 之后的变化

- 新增了 `Turtle.shearfactor()`, `Turtle.shapetransform()` 和 `Turtle.get_shapepoly()` 方法。这样就可以使用所有标准线性变换来调整海龟形状。`Turtle.tiltangle()` 的功能已被加强: 现在可被用来获取或设置倾角。`Turtle.settiltangle()` 已弃用。
- 新增了 `Screen.onkeypress()` 方法作为对 `Screen.onkey()` 的补充，实际就是将行为绑定到 keyrelease 事件。后者相应增加了一个别名: `Screen.onkeyrelease()`。
- 新增了 `Screen.mainloop()` 方法。这样当仅需使用 Screen 和 Turtle 对象时不需要再额外导入 `mainloop()`。
- 新增了两个方法 `Screen.textinput()` 和 `Screen.numinput()`。用来弹出对话框接受输入并分别返回字符串和数值。

- 两个新的示例脚本 `tdemo_nim.py` 和 `tdemo_round_dance.py` 被加入到 `Lib/turtledemo` 目录中。