

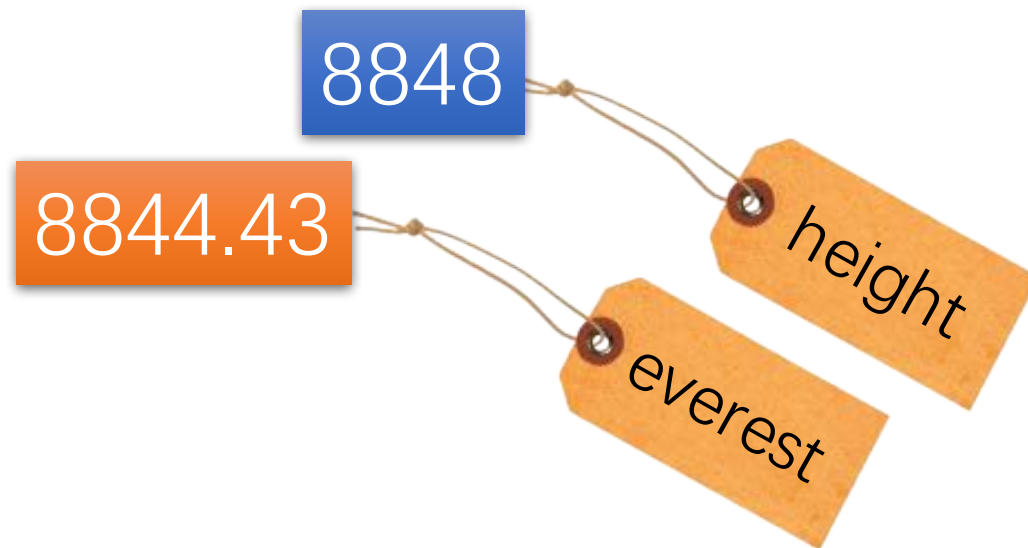
Python程序设计03-基本数据类型

北京大学 陈斌

2019.03.11

目录

- 数值类型，复数类型
- 数据对象和命名
- 海龟🐢作图
- 逻辑类型，字符串类型
- 类型转换
- 输入输出函数
- 写一个完整的Python程序



Python语言的几个要件

数据对象和组织

- 对现实世界实体和概念的抽象
- 分为简单类型和容器类型
- 简单类型用来表示值
 - 整数int、浮点数float、复数complex、逻辑值bool、字符串str
- 容器类型用来组织这些值
 - 列表list、元组tuple、集合set、字典dict
- 数据类型之间几乎都可以转换

赋值和控制流

- 对现实世界处理和过程的抽象
- 分为运算语句和控制流语句
- 运算语句用来实现处理与暂存
 - 表达式计算、函数调用、赋值
- 控制流语句用来组织语句描述过程
 - 顺序、条件分支、循环
- 定义语句也用来组织语句，描述一个包含一系列处理过程的计算单元
 - 函数定义、类定义

Python数据类型：整数int、浮点数float

- 最大的特点是**不限制**大小
- 浮点数受到17位有效数字的限制
- 常见的运算包括加、减、乘、除、整除、求余、幂指数等
- 浮点数的操作也差不多（判断相等要特别注意）
- 一些常用的数学函数如sqrt/sin/cos等都在math模块中
 - import math
 - math.sqrt(2)

```
>>> 5
5
>>> -100
-100
>>> 5 + 8
13
>>> 90 - 10
80
>>> 4 * 7
28
>>> 7 / 2
3.5
>>> 7 // 2
3
>>> 7 % 3
1
>>> 3 ** 4
81
>>> 2 ** 100
1267650600228229401496703205376
>>> divmod(9, 5)
(1, 4)
>>> |
```

整数的进制

进制	表示	例子
十进制decimal	无前缀数字	367
二进制binary	0b前缀	0b101101111
八进制octal	0o前缀	0o557
十六进制hexadecimal	0x前缀	0x16f

- 可以用各种进制表示整数
- 也可以转为字符串
 - `str()`, `bin()`, `oct()`, `hex()`
- 浮点数可以转为十六进制
 - `float.hex()`

```
>>> float.hex(1.23)
'0x1.3ae147ae147aep+0'
>>> (1.23).hex()
'0x1.3ae147ae147aep+0'
```

浮点数的精度问题

- 计算机内部用二进制保存数值,
- 十进制的有限小数转为二进制可能变成无限循环小数
 - $(0.1)_{10} = (0.000110011001\dots)_2$
- 四舍五入将产生误差
- 浮点数判断相等不能简单用相等关系符判断
- 可以视数值取小数点后固定位数进行四舍五入再判断相等

```
>>> 0.2+0.1
0.30000000000000004
>>> 0.2+0.1==0.3
False
>>> round(0.2+0.1, 10)==round(0.3, 10)
True
>>>
```

数值常见的运算和比较

运算符	功能	备注
$m + n$	加法	
$m - n$	减法	
$m * n$	乘法	
$m // n$	整数除法	结果是商的整数部分
m / n	除法	“真”除法，得到小数
$m \% n$	求余数	
<code>divmod(m, n)</code>	求整数除法和余数	会得到两个整数，一个是 $m // n$ ，另一个是 $m \% n$
$m ** n$	求乘方	整数 m 的 n 次方
<code>abs(m)</code>	求绝对值	
$m == n$	相等比较	m 是否等于 n
$m > n$	大于比较	m 是否大于 n
$m >= n$	大于或等于比较	m 是否大于或者等于 n
$m < n$	小于比较	m 是否小于 n
$m <= n$	小于或等于比较	m 是否小于或者等于 n

- 可以进行连续比较判断

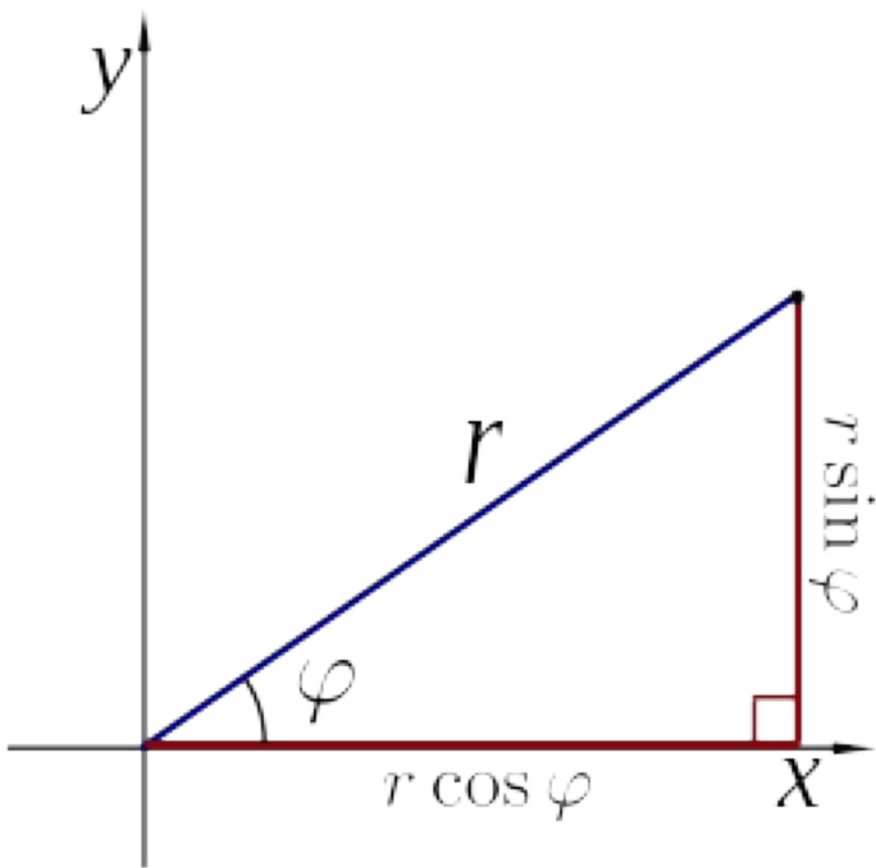
```
>>> 7 > 3 >= 3
True
>>> 12 < 23 < 22
False
>>> m, n = 4, 8
>>> 1 <= m < n <= 10
True
```

Python数据类型：复数

- Python内置复数类型
 - `<class 'complex'>`
- 支持所有常见的复数计算
 - `abs`函数支持复数取模运算
- 对复数处理的数学函数在模块`cmath`中
 - `import cmath`
 - `cmath.sqrt(1+2j)`

```
>>> 1+3j
(1+3j)
>>> (1+2j)*(2+3j)
(-4+7j)
>>> (1+2j)/(2+3j)
(0.6153846153846154+0.07692307692307691j)
>>> (1+2j)**2
(-3+4j)
>>> (1+2j).imag
2.0
>>> (1+2j).real
1.0
>>>
>>> abs(1+2j)
2.23606797749979
```


Python数据类型：复数的形式转换



polar: 极坐标
rect: 直角坐标

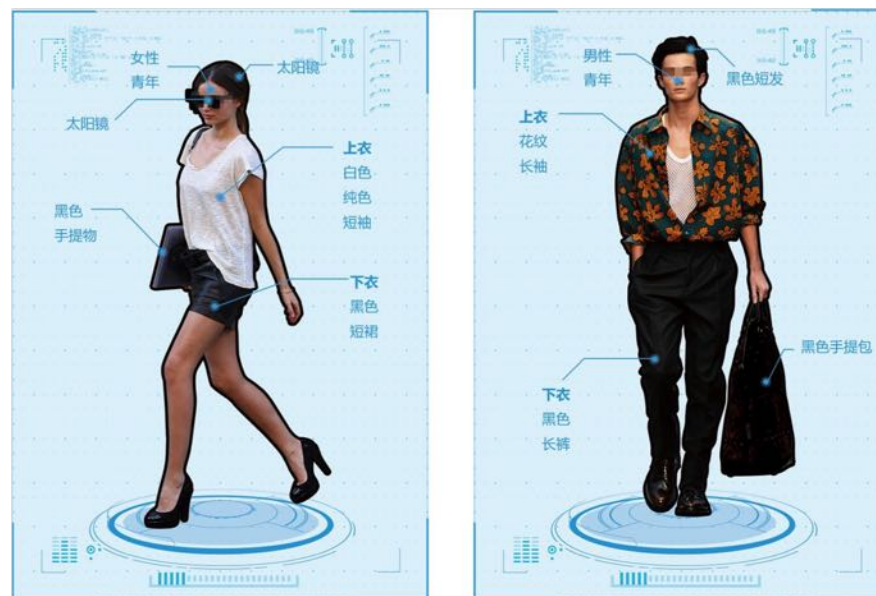
```
>>> import cmath
>>> cmath.polar(3+4j)
(5.0, 0.9272952180016122)
>>> cmath.rect(1, cmath.pi)
(-1+1.2246467991473532e-16j)
>>> |
```

数据对象和命名

- Python语言中几乎所有的事物都是对象（Object）
 - 对象有类型（type）和值（value）
 - 对象有独一无二的标识（id）
 - 对象有一些属性（attribute）
 - 对象还有行为（方法method）

```
>>> type(2019)
<class 'int'>
>>> id(2019)
4452715952
>>> dir(2019)
['_abs_', '_add_', '_and_', '_bool_', '_r_', '_dir_', '_divmod_', '_doc_', '_eq_', '_floordiv_', '_format_', '_ge_', '_getattr_', '_hash_', '_index_', '_init_', '_invert_', '_le_', '_lshift_', '_lt_', '_mod_', '_new_', '_or_', '_pos_', '_pow_', '_reduce_', '_reduce_ex_', '_repr_']
```

- 例如：某个人（object）
 - 人类（type），物质躯体（value）
 - 此人有独特的生理标识（id）
 - 此人有一些特征（attribute）
 - 此人还可以做一些事（method）



给数据对象命名：赋值（assignment）

- 赋值语法

- `<名字> = <数据对象>`

```
>>> n = 1
>>> m = 3.14
>>> msg = "hello"
```

```
>>> OK = 89
>>> me = "chen"
>>> 身高 = 1.85
>>> 1k = 1000
SyntaxError: invalid syntax
```

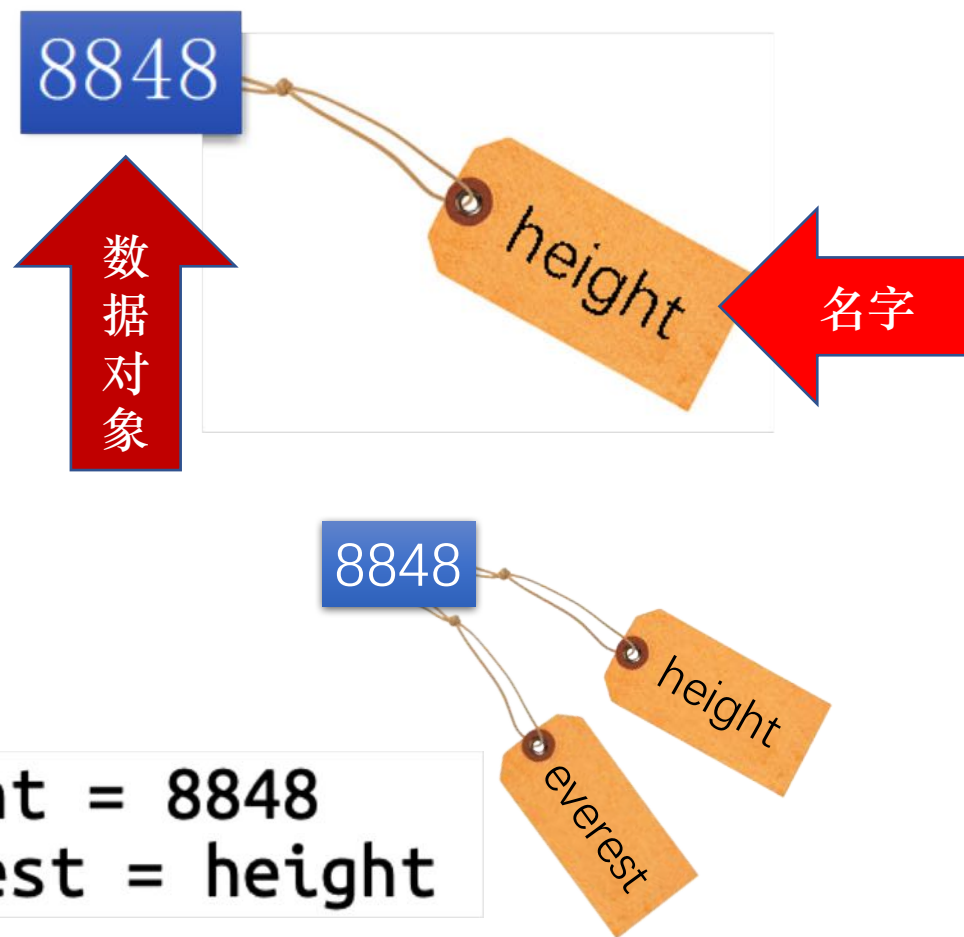
- 名字规则

- **字母**和**数字**组合而成，下划线“_”算字母，字母区分大小写
- **不带特殊字符**（如空格、标点、运算符等）
- 名字的**第一个字符必须是字母**，而不能是数字
- （注：**汉字算是字母**）

- 起名的艺术

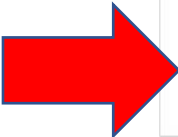
名字 (Name) 与变量 (Variable)

- 名字像一个**标签**，通过赋值来“贴”在某个数据对象上
- 名字和数据对象的关联，称为**引用**。
- 关联数值后的名字，就拥有了数据对象的值 (value)、类型 (type) 和标识 (id)
- 一个数据对象可以和**多个**名字关联

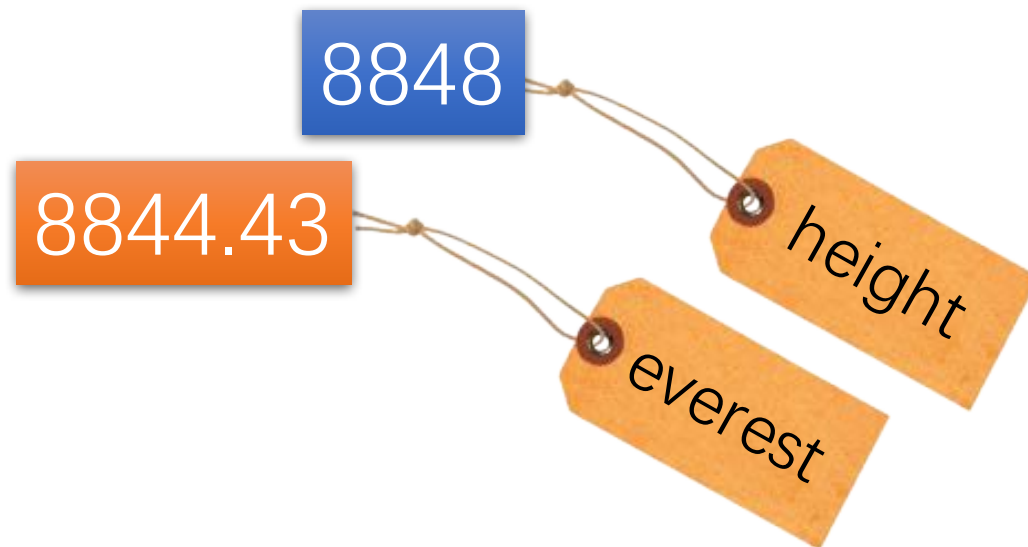


名字 (Name) 与变量 (Variable)

- 与数值关联的名字也称作**变量**, 表示名字的值和类型可以随时**变化**。



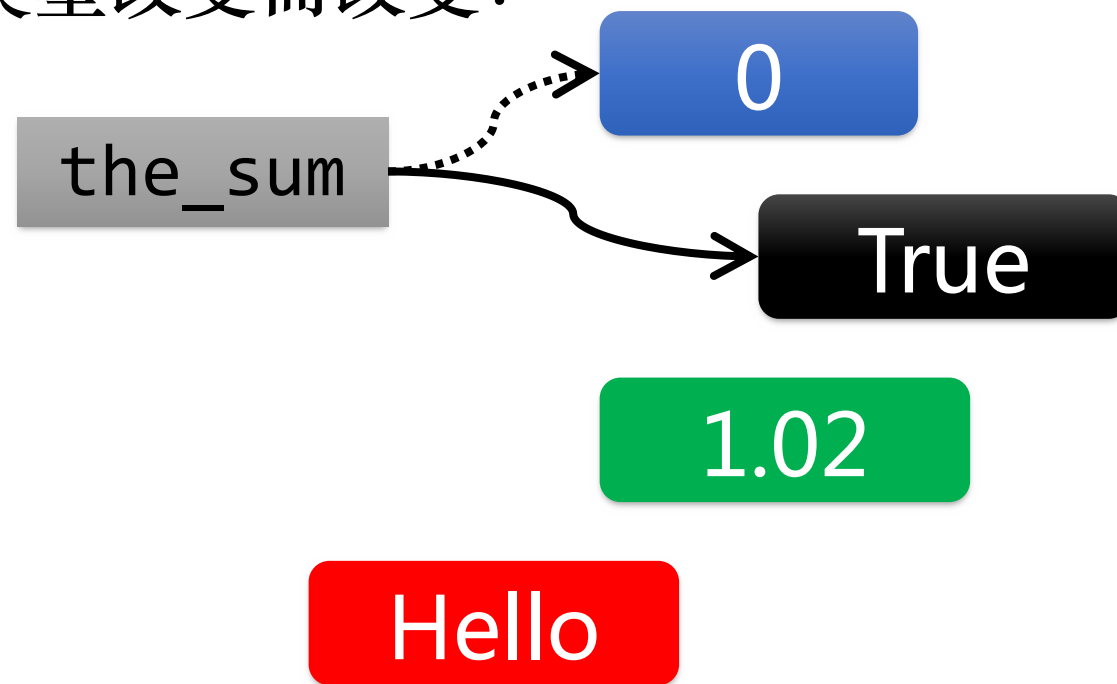
```
height = 8848  
everest = height  
everest = 8844.43
```



名字 (Name) 与变量 (Variable)

- 变量可以随时指向任何一个数据对象
 - 比如True, 1.02, 或者"Hello"
- 变量的类型随着指向的数据对象类型改变而改变!

```
>>> the_sum = 0
>>> type(the_sum)
<class 'int'>
>>> the_sum = True
>>> type(the_sum)
<class 'bool'>
```



灵活多变的赋值语句

- 最基本的赋值语句形式

- `<名字> = <数据对象>`

- 合并赋值

- `a = b = c = 1`

- 按顺序依次赋值

- `a, b, c = 7, 8, 9`

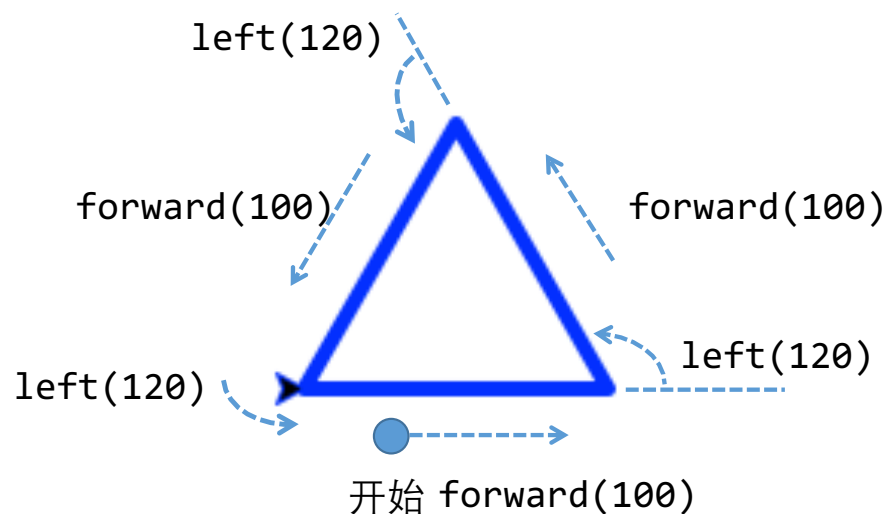
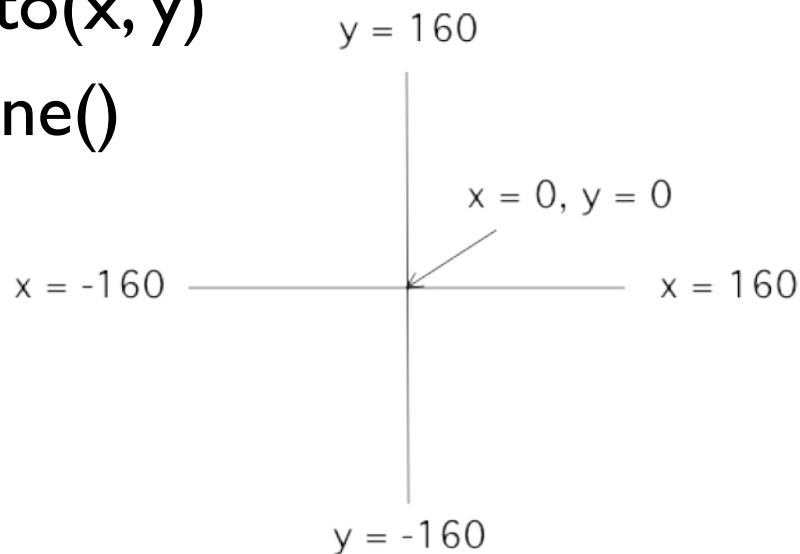
- 简写自操作赋值语句

- `price += 1`
- `price *= 1.5`
- `price /= 3 + 4`

```
>>> a = b = c = 1
>>>
>>> a, b, c = 7, 8, 9
>>>
>>> price = 120
>>> price += 1
>>> price *= 1.5
>>> price /= 2 + 1
```

海龟做图：turtle

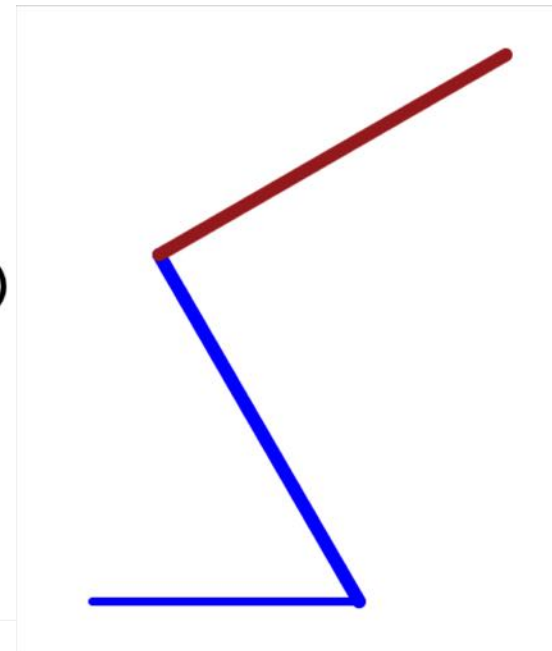
- 模拟海龟在沙滩上爬行所描绘的轨迹，从LOGO语言借鉴而来
 - 前进forward(n)；后退backward(n)；左转left(d)；右转right(d)
- 画笔：抬起落下、颜色、粗细
 - 抬起penup()；落下pendown()；笔色color()；笔粗细pensize(n)
- 直接定位：goto(x, y)
- 结束绘制：done()



作图程序模版

- 首先，导入turtle模块
- 然后，生成一只海龟
 - 可以做一些初始化设定
- 程序主体：用作图语句绘图
- 最后结束作图
 - 可选隐藏海龟： `t.hideturtle()`

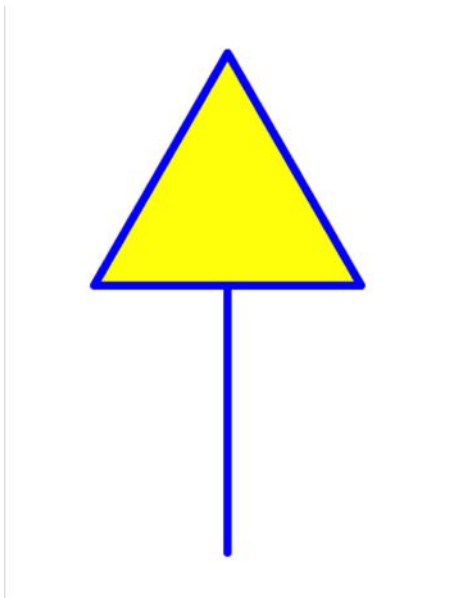
```
1 # 1. 导入海龟模块
2 import turtle
3
4 # 2. 生成一只海龟，做一些设定
5 t = turtle.Turtle()
6 t.color("blue")
7 t.pensize(3)
8
9 # 3. 用海龟作图
10 t.forward(100)
11 t.right(60)
12 t.pensize(5)
13 t.backward(150)
14 t.left(90)
15 t.color("brown")
16 t.forward(150)
17
18 # 4. 结束作图
19 t.hideturtle()
20 turtle.done()
```



样例：警示牌

- 填充fill

- 设定填充颜色fillcolor
- 开始填充begin_fill
- 结束填充end_fill



```
4 # 2. 生成一只海龟，做一些设定
5 t = turtle.Turtle()
6 t.pencolor("blue")
7 t.pensize(3)
8 t.fillcolor("yellow")
9
10 # 3. 用海龟作图
11 t.lt(90)
12 t.fd(100)
13 t.rt(90)
14 t.begin_fill()
15 t.fd(50)
16 t.left(120)
17 t.fd(100)
18 t.left(120)
19 t.fd(100)
20 t.left(120)
21 t.fd(50)
22 t.end_fill()
```

海龟函数的小结

- 前进forward(n)后退backward(n)
 - 缩写: fd(n)、bk(n)
- 左转left(n)、右转right(n)
 - 缩写: lt(n)、rt(n)
- 画笔
 - 笔画颜色pencolor(颜色名称)
 - 笔画粗细pensize(n)
- 抬笔penup()、落笔pendown()
 - 缩写pu()、pd()
- 画圆: circle(半径, 角度)
- 画点: dot(大小, 颜色)
- 填充
 - 填充颜色fillcolor(颜色名称)
 - 填充开始begin_fill()
 - 填充结束end_fill()
- 坐标控制
 - 直接到达goto(x,y)
 - 获取坐标position()
 - 计算距离distance(x,y)

随堂作业：绘制完整的三角警示牌

- 填充：begin_fill, end_fill
- 画圆：circle
- 画点：dot



逻辑类型：bool

- 用来作为判断条件，是逻辑推理的基础
 - 仅有两个值：True、False
- 数值的比较得到逻辑值
 - $3 > 4$
- 逻辑值也有自己的运算
 - and, or, not
- 可以让计算机根据情况自动作出选择，更加聪明

```
>>> type(True)
<class 'bool'>
>>> 1 + 1 == 2
True
>>> 3 > 4
False
>>> n = 5
>>> 1 < n < 10
True
>>> (n > 1) and (n < 10)
True
>>> (1 + 1 == 2) or (3 > 4)
True
>>> not True
False
>>> not (3 > 4)
True
```

判断一个日期是否合法？

- m月d日
- 考虑合法的条件：
- m在1, 3, 5, 7, 8, 10, 12中，同时，d在1~31之间；

```
>>> m in (1,3,5,7,8,10,12) and (1<=d<=31)
```

- 或者，m在4, 6, 9, 11中，同时，d在1~30之间；
- 或者，m是2，d在1~29之间。

随堂作业：判断日期是否合法

- 先写一个逻辑表达式，判断**整数变量y是否闰年**？
- 再结合前面的日期判断
- 例如： $y \% 4 == 0$
- 请继续补充，用and, or
- 可以与邻座同学讨论

字符串类型： str

- 文字字符构成的序列（“串”）
 - 可以表示姓名、手机号、快递地址、菜名、诗歌、小说
- 用双引号或者单引号都可以表示字符串
 - 多行字符串用三个连续单引号表示
- 字符串操作：
 - +连接、*复制、len长度
 - [start:end:step]用来提取一部分（切片slice）

```
>>> 'abc'
'abc'
>>> "abc"
'abc'
>>> '''abc def
ghi jk'''
'abc def\nghi jk'
>>> "Hello\nWorld!"
'Hello\nWorld!'
>>> print("Hello\nWorld!")
Hello
World!
>>> 'abc' + 'def'
'abcdef'|
>>> 'abc' * 4
'ab cab cab cab'
>>> len('abc')
3
>>> 'abcd'[0:2]
'ab'
>>> 'abcd'[0::2]
'ac'
```


练习一下

- 用切片方法，提取出身份证号中的生日

- `c="632622199907012570"`

- 多试几个：

- `350122200101076320`
 - `13062319950406804X`

1
2
3
4
5

`c="632622199907012570"`

`# 13062319950406804X`

`# 350122200101076320`

字符串: str

- 字符来自一个国际标准的大字符集Unicode
- 每种语言的字符都有一个编码
 - 包括表情符号🤗👍
- 可以用函数在编码和字符之间转换
 - chr: 编码到字符
 - ord: 字符到编码



二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形
0010 0000	32	20	(空格) (' ')	0100 0000	64	40	@	0110 0000	96	60	`
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	\$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	'	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k

```
>>> ord("A")
65
>>> ord("中")
20013
```

```
>>> chr(66)
'B'
>>> chr(20012)
'丩'
```

练习一下

- 用chr/ord函数找到你姓名的unicode编码，并且验证一下
- 再多试几个你关心的人名

```
>>> ord("A")  
65  
>>> ord("中")  
20013
```

```
>>> chr(66)  
'B'  
>>> chr(20012)  
'丿'
```

类型转换

- 可以把一个数据对象转换类型，得到新的数据对象
 - "8848", "8844.43": 字符串
 - 8848: 整数
 - 8844.43: 浮点数
- 用类型名称可以直接转换
 - 字符串转数值: int()、float()
 - 数值转字符串: str()、bin()、oct()、hex()

```
>>> int("8848")
8848
>>> float("8844.43")
8844.43
>>> str(8848)
'8848'
>>> bin(8848)
'0b10001010010000'
>>> oct(8848)
'0o21220'
>>> hex(8848)
'0x2290'
```

```
>>> hex(8844.43)
Traceback (most recent call last):
  File "<pyshell#134>", line 1, in <module>
    hex(8844.43)
TypeError: 'float' object cannot be interpreted as an integer
>>> (8844.43).hex()
'0x1.146370a3d70a4p+13'
```

练习一下

- 把前面的姓名unicode编码转换成16进制
- 想一个姓名，把它的16进制unicode编码告诉邻座同学，请TA翻译成中文
 - 都有什么方法？

获取输入：input函数

- 用户给程序的数据在他脑子里，怎么告诉计算机呢？
- input函数通过键盘获取用户输入的字符串
 - 以回车符作为输入结束，一行
- 可以加一个提示符
- 可以把得到的字符串直接转换成其他数据类型

```
>>> x = input("x :")
x :7
>>> y = input("y :")
y :8
>>> x + y
'78'
>>> type(x)
<class 'str'>
>>>
>>> x = int(input("x :"))
x :7
>>> y = int(input("y :"))
y :8
>>> x + y
15
>>> type(x)
<class 'int'>
```


打印输出： print函数

- 计算机把处理结果反馈给用户
- 用print在屏幕上显示数据对象或者变量的值

- `print(v1, v2, v3, ...)`

- 格式化字符串f-strings

- `f"Hello, {name}!"`
 - `f"{name}, you have tried {n} times."`

- 可选的参数

- `sep=" ", end="\n"`

```
>>> name = "Tim"
>>> n = 7
>>> print("Hello", name, "!")
Hello Tim !
>>> print(f"Hello {name}!")
Hello Tim!
>>> print(f"{name}, you have tried {n} times.")
Tim, you have tried 7 times.
>>> print(name, n, sep="#")
Tim#7
```

写一个完整的Python程序

- 导入模块 import
 - 用import导入需要用到的模块;
- 定义函数 def
 - 根据需要定义一批函数;
- 获取数据 input
 - 从键盘输入或者文件读入需要处理的数据;
- 计算处理
 - 按照设计好的算法来进行计算或者处理数据;
- 输出结果 print
 - 将结果输出到屏幕或者写入文件中。

```
# 程序功能:  
# 找到不小于用户输入数的最小质数  
  
# 1, 导入需要的模块  
import math  
  
# 2, 定义函数  
def isprime(n):  
    for i in range(2, int(math.sqrt(n)) + 1):  
        if n % i == 0:  
            return False  
    else:  
        return True  
  
# 3, 获取用户输入的数据  
n = int(input("Please input an integer:"))  
  
# 4, 开始计算搜寻  
temp = n  
while not isprime(temp):  
    temp = temp + 1  
  
# 5, 输出结果  
print("Next prime number is:", temp)
```


随堂作业：计算直角三角形斜边的高

- 写一个完整的程序tc.py
- 要求输入两个直角边长度
 - a, b
- 打印输出斜边上的高h，保留小数点后2位
 - 打印输出如何保留小数点后位数？

```
>>> import math
>>> c = math.sqrt(7)
>>> print(f"c = {c:.5f}")
c = 2.64575
>>> c
2.6457513110645907
```

