Python程序设计

基础知识

刘安
苏州大学，计算机科学与技术学院

http://web.suda.edu.cn/anliu/

本节知识点

- 运算符
- 变量
- 字符串的基本概念
- 函数的基本概念
- 条件语句的基本概念
- math模块的使用

Python IDE

- 目前两个版本Python 3.X和2.X，本课程使用3.X
- 推荐的集成开发环境
  - IDLE3 - Python自带
  - Spyder (Anaconda)
  - Jupyter (Anaconda)
  - PyCharm

算术运算符

```
>>> 3 + 5
8
>>> (3 + 5) * 2 - 1 # 使用括号指定运算优先级
15
>>> 3 + 5 * 2 - 1
12
>>> 6 / 4
1.5
>>> 6 // 4 # 整除，舍弃商的小数部分
1
>>> 10 ** 2 # 幂运算
100
>>> 36 % 10 # 求余运算
6
>>> divmod(36, 10) # 同时计算商和余数
(3, 6)
```

😎 (3, 6)称为元组，元组数据类型后面会详细介绍

关系运算符

- 测试两个对象的大小关系，结果为True或者False
- True表示关系为真，False表示关系为假

```
>>> 3 < 5
True
>>> 3 > 5
False
>>> 3 == 5 # 测试3是否等于5
False
>>> 3 != 5 # 测试3是否不等于5
True
>>> 3 <= 5
True
>>> 3 >= 3
True
```

变量和赋值

- 计算圆的面积 - πr^2

```
>>> 3.14 * (1 ** 2)
3.14
>>>
>>> 3.14 * (10 ** 2)
314.0
```

- 赋值运算符=将变量绑定到值（不是在变量中存储值）

```
>>> pi = 3.14 # pi是一个变量，绑定到值3.14
>>> pi * (1 ** 2)
3.14
>>> pi * (10 ** 2)
314.0
```

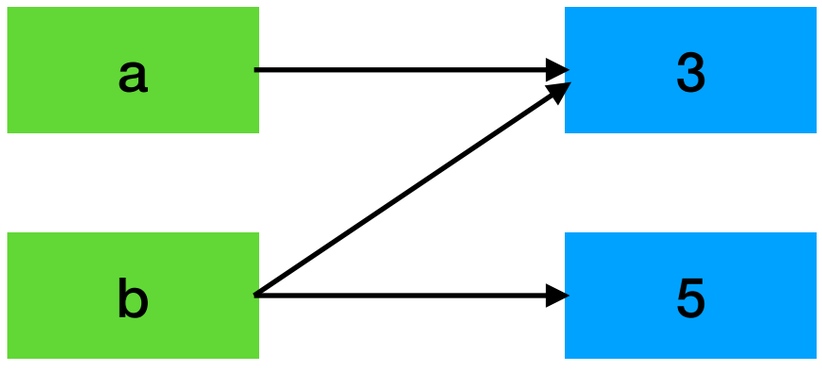
变量和赋值

- 赋值运算符=将变量绑定到值
- 同一个变量绑定的值可以改变

```
>>> pi = 3.14
>>> radius = 1 # 半径
>>> pi * (radius ** 2) # 圆的面积
3.14
>>>
>>> radius = 10 # 变量radius绑定到另一个值10
>>> pi * radius ** 2 # **的优先级高于*
314.0
```

变量、对象和引用

- 创建一个对象（整数3），创建一个变量a
- 赋值运算符将变量a绑定到对象3，或者说变量a引用了对象3
- a+2的值为5，创建一个对象（整数5）
- 将变量b关联到对象5



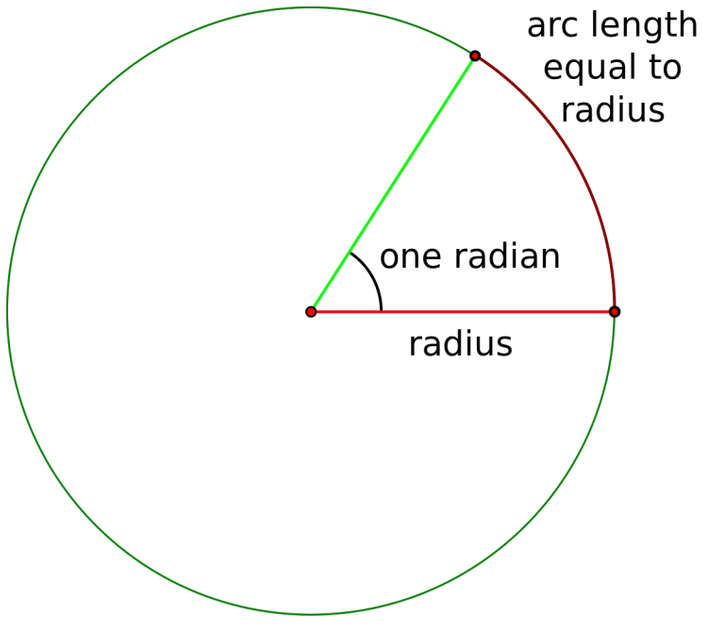

```
>>> id(3)
4431898000
>>> a = 3
>>> id(a)
4431898000
>>> b = 3
>>> id(b)
4431898000
>>> b = a + 2
>>> b
5
>>> id(b)
4431898064
>>> id(5)
4431898064
```

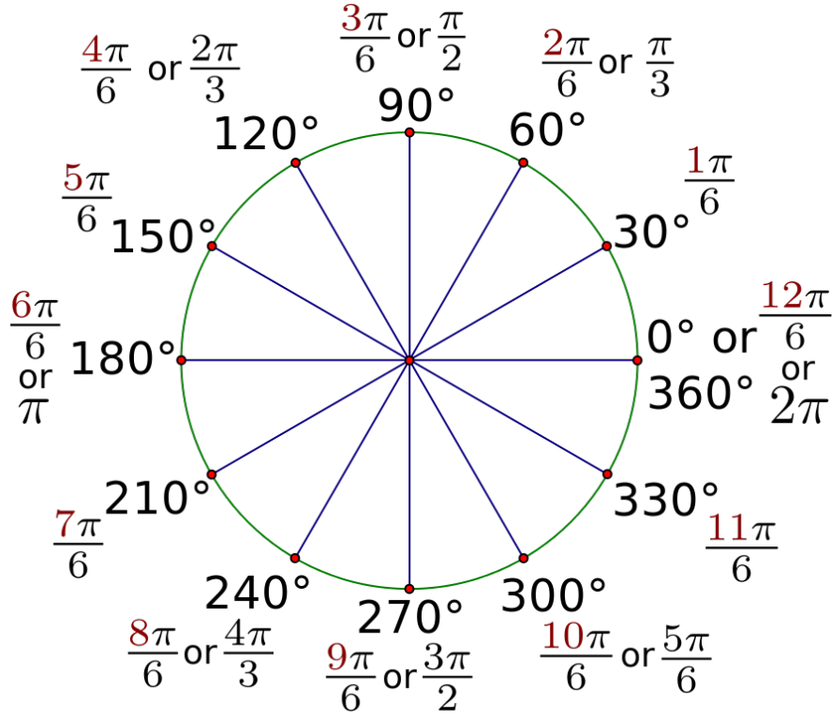
😎 **内置id函数返回对象的标识符，在CPython中，就是对象在内存中的地址**

利用math模块进行复杂的数学运算

- 倍角公式 $\sin(2x) = 2\sin(x)\cos(x)$，x用弧度表示

```
>>> import math # 导入math模块
>>> x = math.pi / 6 # 使用math模块中定义的pi对象
>>> math.sin(2 * x) # 使用math模块中的sin函数
0.8660254037844386
>>> 2 * math.sin(x) * math.cos(x)
0.8660254037844386
```





🧐 **math模块的更多内容参考https://docs.python.org/3/library/math.html**

内置函数dir

- dir(obj)：返回obj的属性

```
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'acos', 'acosh', 'asin'
, 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copy
sign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc
', 'exp', 'expm1', 'fabs', 'factorial', 'floor', '
fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', '
inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ld
exp', 'lgamma', 'log', 'log10', 'log1p', 'log2', '
modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh
', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

内置函数help

- help(obj)：返回obj的帮助信息

```
>>> help(math.factorial)
Help on built-in function factorial in module math:

factorial(...)
    factorial(x) -> Integral

    Find x!. Raise a ValueError if x is negative or
non-integral.

>>> help(math.sqrt)
Help on built-in function sqrt in module math:

sqrt(...)
    sqrt(x)

    Return the square root of x.
```

Four Fours

- 用4个4构造0~50之间的所有整数，仅允许使用下列运算符：加、减、乘、除（不包括整除）、平方、平方根、阶乘、括号()和小数点.

- 类似1.0，2.0这样的数也算整数

```
>>> 44 / 44
1.0
>>> 4 / 4 + 4 / 4
2.0
>>> 4 / .4 + 4 / 4 # .4就是浮点数0.4
11.0
>>> from math import sqrt, factorial #另一种导入方式
>>> (sqrt(4) + factorial(4)) / sqrt(4) + factorial(4)
37.0
```

字符串

- 使用单引号或双引号包围起来的字符序列，用来处理文本

```
>>> name = 'Python' # 单引号
>>> year = "2019" # 双引号
>>> name
'Python'
>>> year
'2019'
>>> len(name) # len函数返回字符串的长度
6
>>> name + year # +连接两个字符串
'Python2019'
>>> year * 3 # *重复字符串，这里重复3次
'201920192019'
```


运算符比如+和*对于不同的对象（整数、字符串）有着不同的含义！

字符串索引操作

- S[i]是字符串S的第i个字符，索引i的有效范围从0到len(S)-1

```
>>> name[0]
'P'
>>> name[1]
'y'
>>> name[5]
'n'
>>> name[6]
Traceback (most recent call last):
  File "<pyshell#542>", line 1, in <module>
    name[6]
IndexError: string index out of range
```

P	y	t	h	o	n
0	1	2	3	4	5

😲 为什么不从1开始编号?

🧐 1972年图灵奖获得者Edsger W. Dijkstra的解释

http://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD831.html

字符串分片操作

- 运算符[i:j]获取字符串的一部分，从第i个字符开始，到第j个字符之前结束（不包括第j个字符），该部分的长度是j-i

H	i		P	y	t	h	o	n	2	0	1	9
0	1	2	3	4	5	6	7	8	9	10	11	12

```
>>> msg = 'Hi' + ' ' + name + year
>>> msg
'Hi Python2019'
>>> msg[0:2]
'Hi'
>>> msg[3:9]
'Python'
>>> msg[9:13] # 注意13不会导致越界错误
'2019'
```

分片操作中省略上下界

- 运算符[i:j]获取字符串的一部分，从第i个字符开始，到第j个字符之前结束（不包括第j个字符），该部分的长度是j-i
- 省略i表示从第0个字符开始
- 省略j表示到最后一个字符结束（包括最后一个）

```
>>> name[0:2]
'Py'
>>> name[2:6]
'thon'
>>> name[ :2]
'Py'
>>> name[2: ]
'thon'
```

```
>>> name[:1] + name[1:]
'Python'
>>> name[:2] + name[2:]
'Python'
>>> name[:3] + name[3:]
'Python'
```

注意 - 关于整数

- Python 3.x中整数可以是任意长度（只要有足够多的内存）

```
>>> 2019 ** 50 # 3.x中整数可以是任意长度
180640841280567853016613273134136943340652054673367388964282268906684193334689794470936642084400304647484512490891664968113712908110682601629876467326313340661758900 1
>>>
>>> len(str(2019**50)) # str函数将整数转换成字符串
166
>>>
>>> len(str(2019**2019))
6674
```

注意 - 关于浮点数

- 浮点数在内存中总是以近似值存储的
- 所以不能直接通过==判断两个浮点数是否相等

```
>>> a = 0.1 + 0.1 + 0.1 # 只能存储浮点数的近似值
>>> b = 0.3             # 只能存储浮点数的近似值
>>>
>>> a == b # 注意a和b相等吗?
False
>>>
>>> # 将两者差的绝对值和一个很小的数进行比较
>>> abs(a-b) < 1e-10
True
```

注意 - 关于字符串

- 引号并不是字符串的一部分，仅仅是用来表示一个字符串
- 数字3和字符串3并不是一回事

```
>>> len('python')
6
>>> len('')
0
>>> 3 + 5
8
>>> '3' + '5'
'35'
```

函数

- 函数可以看成一个黑盒，给定一个输入，函数返回一个值


```
>>> def f(x): # 定义函数，f是函数名，x是参数
        return 3 * x + 1 # 3*x+1是返回值

>>> f(0) # 调用函数，将值0赋给参数x，返回值是1
1
>>> f(1) # 再次调用函数，将值1赋给参数x，返回值是4
4
```

对于一个输入x，数学中的函数必须有唯一确定的输出
对于一个输入x，编程中的函数（每次调用）可以返回（输出）不同的值

函数

```
>>> def f(x): # 定义函数，f是函数名，x是参数
        return 3 * x + 1 # 3*x+1是返回值

>>> f(0) # 调用函数，将值0赋给参数x，返回值是1
1
>>> f(1) # 再次调用函数，将值1赋给参数x，返回值是4
4
```

- 关键字def表示函数定义的开始，其后是自定义的函数名
- 参数放在函数名后小括号内，最后的冒号不可缺少
- 冒号下方的语句需要缩进（比如一个制表符）
- return 语句表示函数停止执行，并将值返回给调用者

同一输入不同输出的函数

```
>>> import random
>>> help(random.random)
Help on built-in function random:

random(...) method of random.Random instance
    random() -> x in the interval [0, 1).
```

```
>>> def g(x):
        return x + random.random()

>>> g(1)
1.2454783285671198
>>> g(1)
1.1145206261271141
```

random模块的更多内容参考https://docs.python.org/3/library/random.html

具有多个参数的函数

- 函数可以具有多个参数，即具有多个输入
- 编写函数perfect_square，接受两个实数x和y，返回值 $x^2 + 2xy + y^2$

```
1 def perfect_square(x, y):
2     total = x ** 2
3     total = total + 2 * x * y
4     total = total + y ** 2
5     return total
```

```
>>> perfect_square(4, 6)
100
>>> perfect_square(19, 41)
3600
```

代码块与控制流

- 第2~5行属于一个代码块，一般情况下这些语句是顺序执行的

```
1 def perfect_square(x, y):
2     total = x ** 2
3     total = total + 2 * x * y
4     total = total + y ** 2
5     return total
```

- 通过缩进来确定哪些语句在同一个代码块中
- 缩进就是语句左侧的空白，没有规定缩进需要多少空白，常见的缩进是1个制表符或者4个空格，但不要混合使用
- 具有同样缩进的语句属于同一个代码块

problem.py

problem.py

```python
def perfect_square(x, y):
    total = x ** 2
    total = total + 2 * x * y
    total = total + y ** 2
    return total
```

Line 9, Column 5 UTF-8 Tab Size: 4 Python

代码 — -bash — 52×7

```
AndeMacBook-Pro-3:代码  ryan$ python problem.py
  File "problem.py", line 3
    total = total + 2 * x * y
                            ^
IndentationError: unindent does not match any outer
indentation level
AndeMacBook-Pro-3:代码  ryan$
```

第2行用的是一个制表符，第3-5行用的是4个空格

混用制表符和空格，结果导致了错误！

考拉兹猜想

- 对于任何一个正整数，如果它是奇数，对它乘3再加1，如果它是偶数，对它除以2，如此循环，最终都能够得到1

```
13  40  20  10  5  16  8  4  2  1
```

- 编写一个函数collatz，接受一个正整数n
  - 如果n是奇数，返回3*n+1：return 3*n + 1
  - 如果n是偶数，返回n//2：return n//2
  - 如何让语句在特定条件下执行？

仅有if分支的条件语句

- 让语句在特定条件下执行
- if condition + 冒号 + 代码块（需要缩进）

```
1 x = 1
2 if condition:
3     #code block
4     #code block
5 y = 1
```

冒号不可少

需要缩进

- 如果第2行中的condition为真，执行if分支对应的代码块（上图中是3~4行），执行完毕后执行if条件语句后面的语句（上图中是第5行）
- 否则直接跳过if分支对应的代码块（上图中是3~4行），直接执行if条件语句后面的语句（上图中是第5行）

if-else条件语句

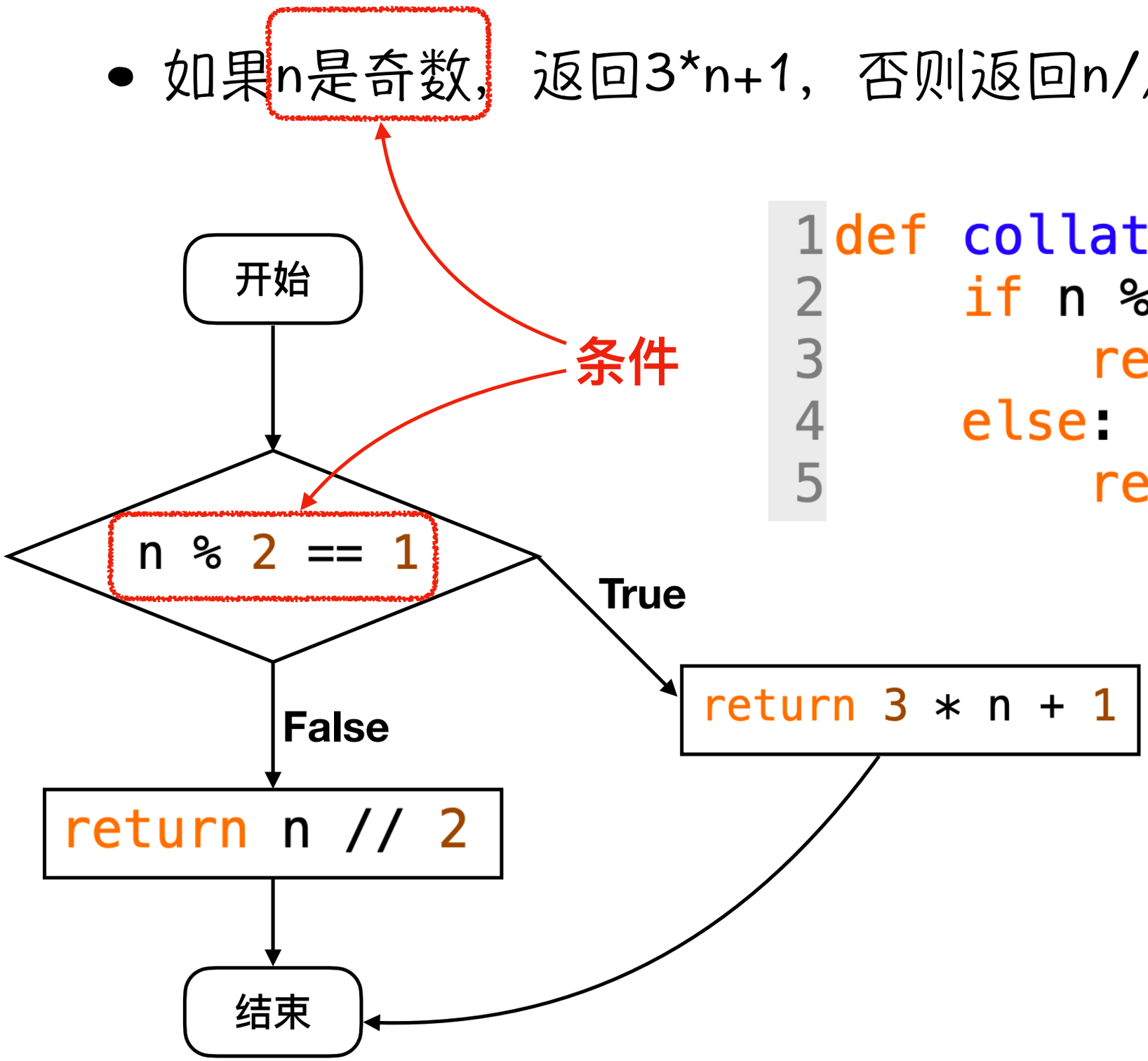
- if condition + 冒号 + 代码块（需要缩进）
  else + 冒号 + 代码块（需要缩进）

```
1 x = 1
2 if condition:
3     #code block a
4 else:
5     #code block b
6 y = 1
```

- 如果第2行中的condition为真，执行if分支对应的代码块a，执行完毕后执行if条件语句后面的语句（上图中是第6行）
- 否则直接跳过if分支对应的代码块a，执行else分支对应的代码块b，然后执行if条件语句后面的语句（上图中是第6行）

利用条件语句实现考拉兹函数

- 如果n是奇数，返回3*n+1，否则返回n//2




```
1 def collatz(n):
2     if n % 2 == 1:
3         return 3 * n + 1
4     else:
5         return n // 2
```

复合语句与缩进

- 复合语句 = 首行 + 冒号 + 代码块（需要缩进）

- def语句（定义函数）

```
1 def perfect_square(x, y):
2     total = x ** 2
3     total = total + 2 * x * y
4     total = total + y ** 2
5     return total
```

- 条件语句（根据条件执行特定的语句）

```
1 def collatz(n):
2     if n % 2 == 1:
3         return 3 * n + 1
4     else:
5         return n // 2
```

```
1 def collatz(n):
2     if n % 2 == 1:
3         return 3 * n + 1
4     else:
5         return n // 2
```

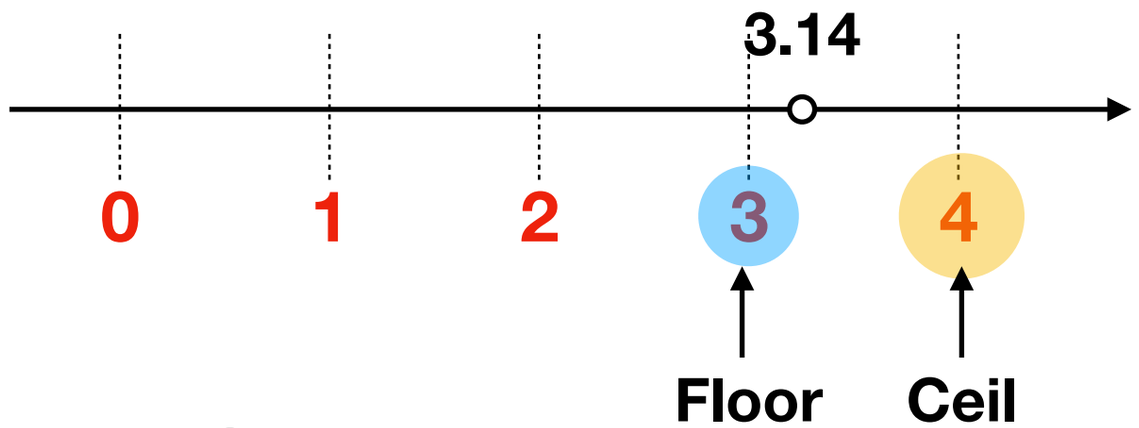
- def语句：首行L1，代码块 (只有一个if-else条件语句，需要缩进，注意L2相对于L1的位置)
- if-else条件语句：有多个首行(L2和L4)，需要相同的缩进
  - 首行2 (if分支)，代码块只有语句3，需要缩进，注意L3相对于L2的位置
  - 首行4 (else分支)，代码块只有语句5，需要缩进，注意L5相对于L4的位置
  - 注意：L3和L5属于不同的代码块，所以缩进可以不同

```
1 def func_show_indent():
2     s = 'block_a'
3     x = 1
4     if x % 2 == 1:
5         y = 2
6         if y % 2 == 1:
7             s = 'block_c'
8         s = 'block_b'
9     return s
```

- def语句1：代码块包含语句2、3、4、9，它们具有相同的缩进，注意语句4是一个复合语句
- 复合语句4：首行4，代码块包含语句5、6、8，它们具有相同的缩进，注意语句6又是一个复合语句
- 复合语句6：首行6，代码块只有语句7

函数Floor和Ceil

- 函数floor(x)返回小于等于x的最大整数，也记为$\lfloor x \rfloor$
- 函数ceil(x)返回大于等于x的最小整数，也记为$\lceil x \rceil$

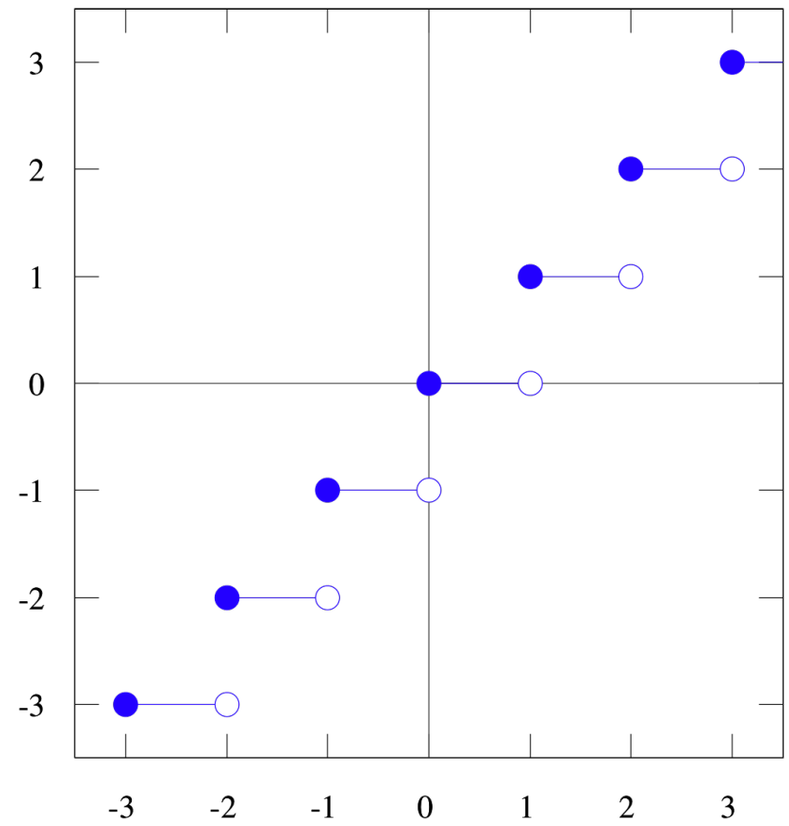



```
>>> import math
>>> math.floor(3.14)
3
>>> math.ceil(3.14)
4
```

if-elif-else条件语句

- 1个if分支，后跟任意数量的elif分支（包括0个），后跟1个或者0个else分支，每个分支有一个对应的代码块
- 从上向下依次判断条件，如果为真，执行对应的代码块，否则判断下一个条件

```
1 def my_floor(x):
2     if x < 0:
3         return math.floor(x)
4     elif x < 1: # 0 <= x < 1
5         return 0
6     elif x < 2: # 1 <= x < 2
7         return 1
8     else: # x >= 2
9         return math.floor(x)
```



字符串作为参数的函数

- 函数的参数可以是数字、也可以是字符串等其他类型
- 下面的magic函数具有什么样的功能？

```
1 def magic(first, second):
2     a = first[1:]
3     b = second[len(second) - 1]
4     return a + b
```

```
>>> magic('hello', 'python')
'ellon'
```

简单求和

- 编写一个函数，接收一个正整数n，返回1~n的和
  $S = 1 + 2 + \cdots + n$

  - $S = 1 + 2 + \cdots + n = n(n+1)/2$

```
>>> def sum_of_n(n):
        return n * (n+1) // 2

>>> sum_of_n(5)
15
>>>
>>> sum_of_n(100)
5050
```

平方和

- 编写一个函数，接收一个正整数n，返回1~n的平方和
  $S = 1^2 + 2^2 + \cdots + n^2$

  - $S = 1^2 + 2^2 + \cdots + n^2 = \dfrac{n(n+1)(2n+1)}{6}$

```
>>> def sum_of_n2(n):
        return n*(n+1)*(2*n+1) // 6

>>> sum_of_n2(3)
14
>>>
>>> sum_of_n2(100)
338350
```

个十百位的数字和

- 编写一个函数，接收一个整数，返回其个位、十位和百位数字的和

```
>>> def sum_of_one_ten_hund(n):
        ones = n % 10
        n = n // 10
        tens = n % 10
        n = n // 10
        hunds = n % 10
        return ones + tens + hunds

>>> sum_of_one_ten_hund(123)
6
>>> sum_of_one_ten_hund(12)
3
>>> sum_of_one_ten_hund(1234)
9
```

整除

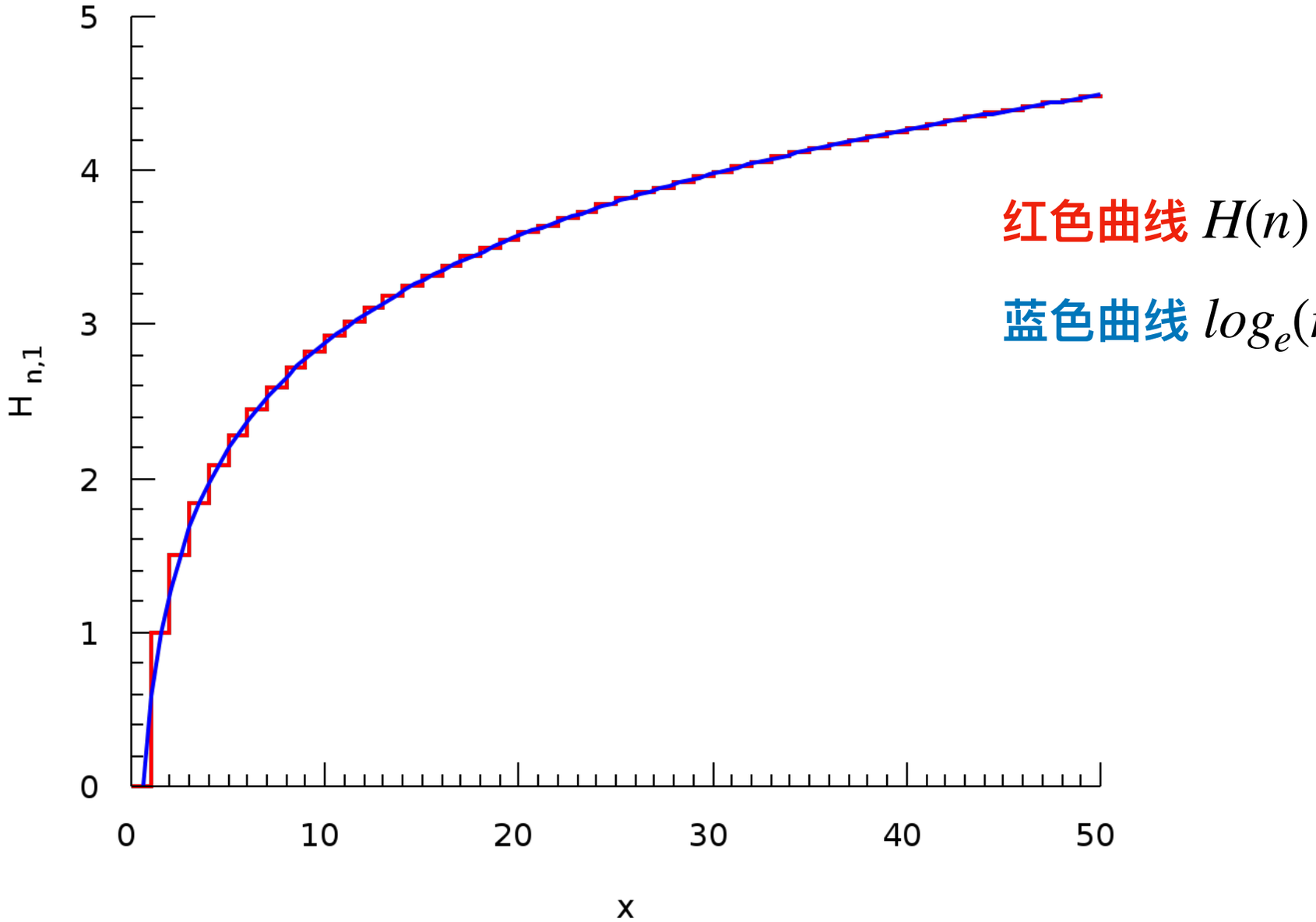
- 编写一个函数，接收2个正整数m和n，如果其中一个整数能被另一个整数整除，返回True，否则返回False

```
>>> def one_divide_another(m, n):
        if m % n == 0:
            return True
        elif n % m == 0:
            return True
        else:
            return False


>>> one_divide_another(5, 3)
False
>>> one_divide_another(5, 30)
True
```

调和数

- 调和数 $H(n) = 1 + 1/2 + 1/3 + \cdots + 1/n$



调和数

- 调和数 $H(n) = 1 + 1/2 + 1/3 + \cdots + 1/n$
- 编写一个函数，接受一个正整数 n，返回调和数 $H(n)$
  - 如果 $n < 100$，调用函数harmonic_small计算 $H(n)$
  - 如果 $n >= 100$，编写函数harmonic_large计算$H(n)$，该函数使用下面的近似公式进行计算
    - $H(n) = log_e(n) + \gamma + 1/(2n) - 1/(12n^2) + 1/(120n^4)$
    - 其中 $\gamma = 0.5772156649015 32$ 称为欧拉常数

调和数$H(n) = 1 + 1/2 + 1/3 + \cdots + 1/n$

- 函数harmonic_small( )的实现

```
>>> def harmonic_small(n):
        total = 0 # 存放H(n)的值
        for i in range(1, n+1): # i依次等于1,2,...,n
            total = total + 1 / i # 把项1/i计入总和中
        return total
```

- for语句也是复合语句，其对应的代码块是什么？

- for语句用来重复执行其对应的代码块

```
>>> harmonic_small(2)
1.5
>>> harmonic_small(3)
1.8333333333333333
>>> harmonic_small(4)
2.083333333333333
```

调和数$H(n) = 1 + 1/2 + 1/3 + \cdots + 1/n$

- 函数harmonic_large( )的实现

$$H(n) = log_e(n) + \gamma + 1/(2n) - 1/(12n^2) + 1/(120n^4)$$

```
>>> import math
>>>
>>> def harmonic_large(n):
    gamma = 0.577215664901532
    return (math.log(n, math.e)
            + gamma
            + 1 / (2 * n)
            - 1 / (12 * n ** 2)
            + 1 / (120 * math.pow(n, 4))
            )
```

将语句放在( )中，就可以跨越多行

调和数$H(n) = 1 + 1/2 + 1/3 + \cdots + 1/n$

- 函数harmonic没有直接计算 $H(n)$
- 根据 n 的大小调用不同的函数来计算 $H(n)$

```
>>> def harmonic(n):
        if n < 100:
            return harmonic_small(n)
        else:
            return harmonic_large(n)
```

```
>>> harmonic(10000)
9.787606036044382
>>>
>>> harmonic_small(10000)
9.787606036044348
```