# Python基础语法精讲

嵩 天

# 基本数据类型

嵩天

基本数据类型

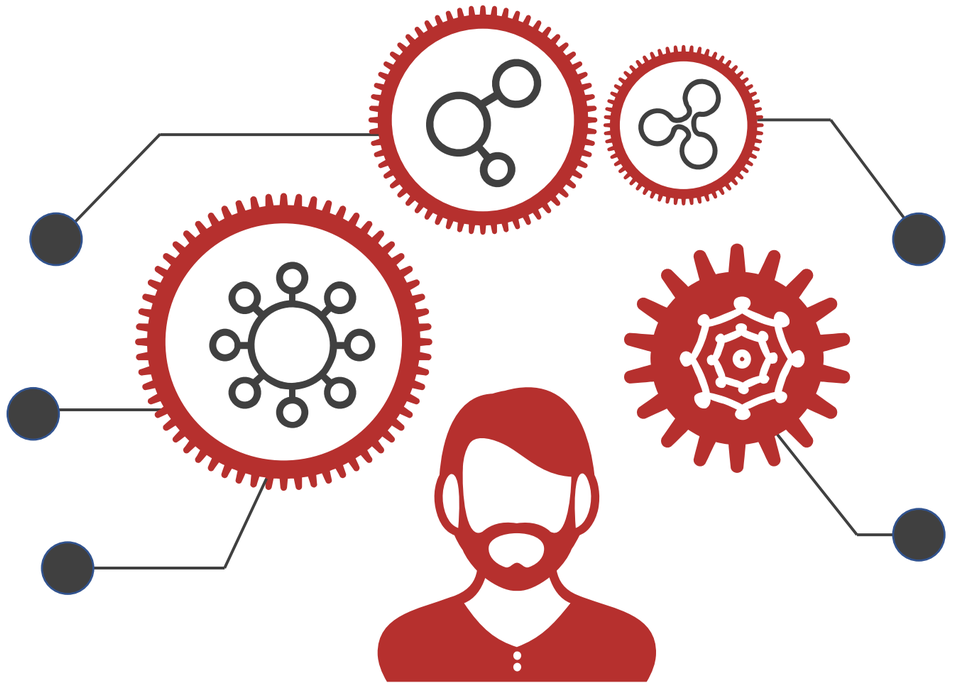
# 单元开篇

# 单元开篇

(1) 数值类型

(2) 数值运算函数与方法

(3) 整数的位运算



(4) 字符串类型

(5) 字符串操作函数与方法

(6) 字节串类型

(7) 字节串操作函数与方法

## 基本数据类型

# 数据类型

## Python语言包括9种基本数据类型

- 数值类型：整数、浮点数、复数
- 字节类型：字符串、字节串
- 组合类型：集合、元组、列表、字典

基本数据类型

# 数值类型

# 数字类型

## 数值类型包括：整数、浮点数和复数

- 整数类型：无取值范围、二进制/八进制/十进制/十六进制
- 浮点数类型：有取值范围、不确定尾数问题、科学计数法、大精确浮点运算
- 复数类型：与数学中复数概念一致、获取实部和虚部

# 整数类型

## 取值范围

- 可正可负，没有取值范围

```
print(pow(2, pow(2, 10)))
```

```
179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216
```

# 整数类型

4种进制表示形式

1. 十进制：123, -321, 0
2. 二进制：以0b或0B开头：0b1101, -0B10
3. 八进制：以0o或0O开头：0o456, -0O789
4. 十六进制：以0x或0X开头：0x1A, -0X2B

# 浮点数类型

## 浮点数的科学计数法表示

使用字母e或E作为幂的符号，以10为基数，格式如下：

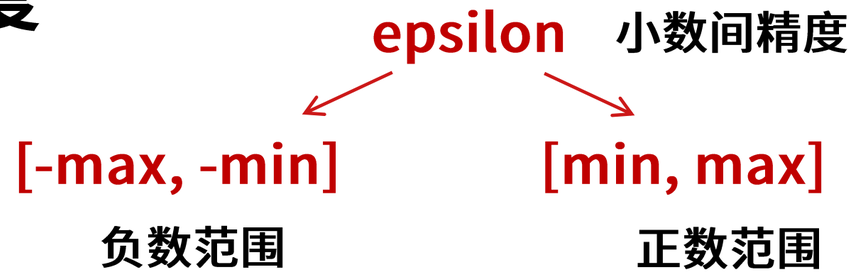
<a>e<b>　　表示 $a*10^b$

例如：1.2e-3值为0.0012　9.8E7值为98000000.0

# 浮点数类型

## 范围和精度

- 有取值范围和精度约定




```
import sys
print(sys.float_info)
```

```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024,
max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021,
min_10_exp=-307, dig=15, mant_dig=53,
epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

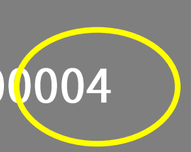
# 浮点数类型

## 不确定尾数问题

- 浮点数直接运算，可能产生不确定尾数

```
print(0.1 + 0.2)
```

```
0.30000000000000004
```



# 浮点数类型

## 不确定尾数问题

0.1

0.0001100110011001100110011001100110011001100110011001100110011010 (二进制)

0.1000000000000000055511151231257827021181583404541015625 (十进制)

不确定尾数问题来源于浮点数在计算机中表示不精确的实际情况，广泛存在于编程语言中

# 浮点数类型

## 不确定尾数问题

- 使用round()辅助浮点数运算，消除不确定尾数

```
round(0.1 + 0.2, 1)
```

```
0.3
```

# 浮点数类型

## 不确定尾数问题

- round(x, d)：对x四舍五入，d是小数截取位数
- 浮点数间运算及比较用round()函数辅助
- 不确定尾数一般发生在$10^{-16}$左右，round()十分有效

# 浮点数类型

## 大精确浮点数运算

- 需求：浮点数的科学运算、精度需求更多、数据运算范围更大
- 解决方案：
  - 将浮点数运算转换为整数运算：数值整数 与 小数位数整数

  例如：1.2e-3表示为 12 和 4，0.01表示为 1 和 2

# 浮点数类型

## 大精确浮点数运算

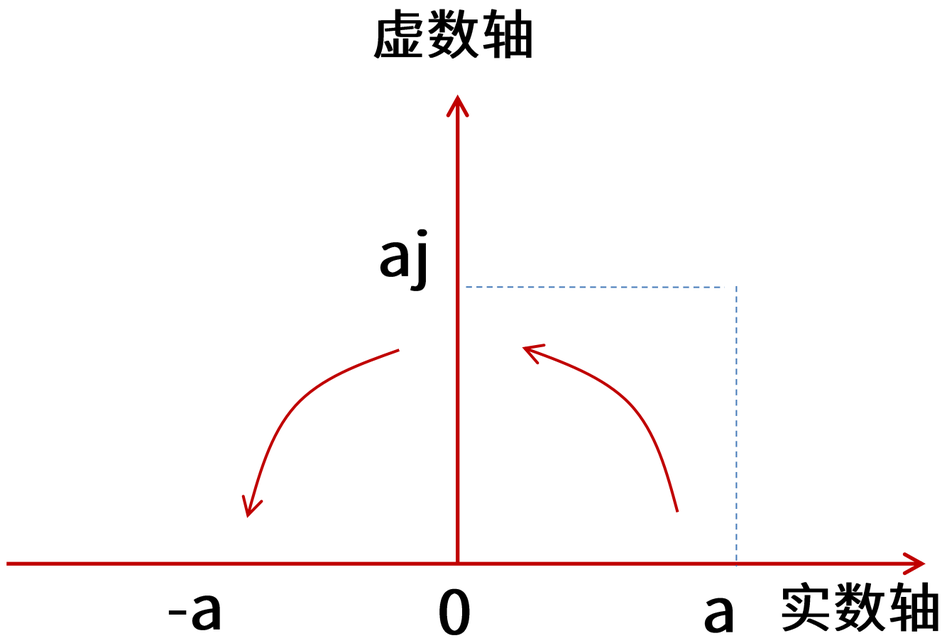
1.2e-3 + 0.01

- 经过转换为 12, 4 + 1, 2
- 将小数位数对齐 12, 4 + 100, 4
- 运算结果 112, 4 即 0.0112

# 复数类型

## 数学中复数概念的直接表示

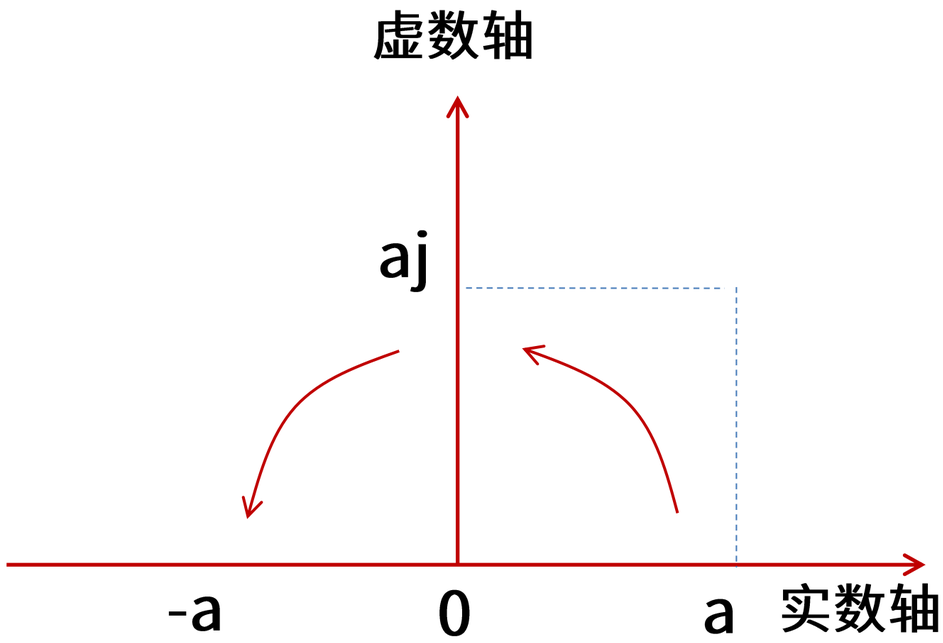
- 定义 $j = \sqrt{-1}$，复数表示为 a + bj
- 其中，a和b都是浮点数



# 复数类型

## 获取复数的实部和虚部

- z = a + bj
- z.real  获得复数z的实部
- z.imag 获得复数z的虚部



# 小结：数值类型

**数值类型包括：整数、浮点数和复数**

- 整数类型：无取值范围、二进制/八进制/十进制/十六进制
- 浮点数类型：有取值范围、不确定尾数问题、科学计数法、大精确浮点运算
- 复数类型：与数学中复数概念一致、获取实部和虚部

基本数据类型

# 数值运算函数与方法

Python 123

# 数值运算

## 数值运算：操作符、函数和方法

- 操作符：+　-　*　/　// 等等
- 函数：Python解释器提供的内置函数
- 方法：数值类型在Python解释器内部都是类(class)，类的方法

# 数值运算操作符

| 操作符 | 描述 |
| --- | --- |
| x + y | 加，x与y之和 |
| x – y | 减，x与y之差 |
| x * y | 乘，x与y之积 |
| x / y | 除，x与y之商  10/3结果是3.3333333333333335 |
| x // y | 整数除，x与y之整数商  10//3结果是3 |
| + x | x本身 |
| - y | x的负值 |
| x % y | 余数，模运算  10%3结果是1 |
| x ** y | 幂运算，x的y次幂，$x^y$ |

# 赋值增强操作符

| 赋值增强操作符 | 描述 |
| --- | --- |
| x op= y | 即 x = x op y，其中，op为二元操作符 |
| | x += y　x -= y　x *= y　x /= y<br>x //= y　x %= y　x **= y |
| | 常用实例：<br>变量加1操作：n += 1<br>变量减k操作：n -= k<br>Python没有x++操作 |

# 数值类型的运算关系

**类型间可进行混合运算，生成结果为"最宽"类型**

- 三种类型存在一种逐渐"扩展"或"变宽"的关系：

  整数 -> 浮点数 -> 复数

- 例如：123 + 4.0 = 127.0 (整数+浮点数 = 浮点数)

# 数值运算函数

| 函数及使用 | 描述 |
| --- | --- |
| abs(x) | 绝对值，x的绝对值 |
| divmod(x,y) | 商余，(x//y, x%y)，同时输出商和余数 |
| pow(x, y[, z]) | 幂余，(x**y)%z，[..]表示参数z可省略 |
| round(x[, d]) | 四舍五入，d是保留小数位数，默认值为0 |
| max($x_1$,$x_2$, ⋯ ,$x_n$) | 最大值，返回$x_1$,$x_2$, ⋯ ,$x_n$中的最大值，n不限 |
| min($x_1$,$x_2$, ⋯ ,$x_n$) | 最小值，返回$x_1$,$x_2$, ⋯ ,$x_n$中的最小值，n不限 |

# 数值运算函数

| 函数及使用 | 描述 |
| --- | --- |
| int(x) | 将x变成整数，舍弃小数部分 |
| float(x) | 将x变成浮点数，增加小数部分 |
| complex(x) | 将x变成复数，增加虚数部分 |

基本数据类型

# 整数的位运算

Python▸123

# 整数的位运算

## 整数之间可以进行位运算

- 位运算按照二进制方式逐位进行
- 位运算只针对整数有作用

# 位运算符

| 位运算符 | 描述 |
| --- | --- |
| x & y | 与，x与y的与操作 |
| x \| y | 或，x与y的或操作 |
| ~x | 非，x按位取反 |
| x ^ y | 异或，x与y的异或操作 |
| x << n | 左移，将x按位左移n位 |
| x >> n | 右移，将x按位右移n位 |

# 位运算符实例

```
>>>101 & 99
97
>>>bin(101)
'0b1100101'
>>>bin(99)
'0b1100011'
>>>0b1100001
97
```

```
>>>~101
-102
>>>bin(-102)
'-0b1100110'
>>> bin(101>>4 )
'0b110'
>>>bin(101<<4 )
'0b11001010000'
```

101二进制　'0b1100101'

101二进制　000…0001100101

101取反　　111…1110011010

101取反，首位是1，负数的补码

**补码->源码：取反+1**

101取反二进制　'-0b1100110'

**(切记：这只是表象)**

基本数据类型

# 字符串类型

Python 123

# 字符串

## 字符串：由0个或多个字符组成的有序字符序列

- 单行字符串由一对单引号或一对双引号表示

  "〇一二三四五六七八九十"

- 多行字符串由三个单引号或三个双引号表示

  ''' Python

  语言 '''

Q: 单双引号作用一样吗？为何提供2种

Q: 三个单引号不是注释吗？

# 字符串的序号

# 字符串的索引和切片

## 使用[ ]获取字符串中一个或多个字符

- 索引：返回字符串中单个字符 <字符串>[M]

  "〇一二三四五六七八九十"[1]　　结果是 "一"

- 切片：返回字符串中一段字符子串 <字符串>[M: N] 不含N

  "〇一二三四五六七八九十"[1: 5]　结果是 "一二三四"

# 字符串切片的高级用法

## 使用[M: N: K]根据步长对字符串切片

- <字符串>[M: N]，M缺失表示至开头，N缺失表示至结尾

  "〇一二三四五六七八九十"[:3]　　结果是 "〇一二"

- <字符串>[M: N: K]，根据步长K对字符串切片

  "〇一二三四五六七八九十"[1: 8: 2]　结果是 "一三五七"

  "〇一二三四五六七八九十"[::-1]　　结果是 "十九八七六五四三二一〇"

# 字符串的特殊字符：转义符 \

## 转义符是表达特殊字符串或功能的方式

- 转义符表达特定字符的本意

  "这里有个双引号(\")" 结果为 这里有个双引号(")

- 转义符形成一些组合，表达一些不可打印的含义

  "\b"回退 "\n"换行(光标移动到下行首) "\r" 回车(光标移动到本行首)

# 字符串操作符

| 操作符及使用 | 描述 |
| --- | --- |
| x + y | 连接两个字符串x和y |
| n * x 或 x * n | 复制n次字符串x |
| x in s | 如果x是s的子串，返回True，否则返回False |

# 字符串操作符

## 获取星期字符串

- 输入：1-7的整数，表示星期几
- 输出：输入整数对应的星期字符串
- 例如：输入3，输出 星期三

```
weekStr = "星期一星期二星期三星期四星期五星期六星期日"
weekId = eval(input("请输入星期数字(1-7)："))
pos = (weekId - 1) * 3
print(weekStr[pos: pos+3])
```

```
weekStr = "一二三四五六日"
weekId = eval(input("请输入星期数字(1-7)："))
print("星期" + weekStr[weekId-1])
```

基本数据类型

# 字符串操作函数与方法

# 字符串处理函数

## 一些以函数形式提供的字符串处理功能

| 函数及使用 | 描述 |
| --- | --- |
| len(x) | 长度，返回字符串x的长度 |
| str(x) | 任意类型x所对应的字符串形式 |
| hex(x) 或 oct(x) | 整数x的十六进制或八进制小写形式字符串 |
| chr(u) | x为Unicode编码，返回其对应的字符　（源于：character） |
| ord(x) | x为字符，返回其对应的Unicode编码　（源于：ordinal） |

# Unicode编码

## Python字符串的编码方式

- 统一字符编码，即覆盖几乎所有字符的编码方式
- 从0到1114111 (0x10FFFF)，每个编码对应一个字符
- Python字符串中每个字符都是Unicode编码字符

```
>>> "1 + 1 = 2" +  chr(10004)
'1 + 1 = 2✔'
>>>ord('♉')
9801
```

# 字符串处理方法

“方法”是一个面向对象中的专有名词

- 方法特指类中的函数，调用时表现为<a>.<b>()方式
- 方法本身也是函数，即<b>()，但与<a>有关
- Python所有类型本质上都是类，即<a>，存在处理方法

# 字符串处理方法

| 方法及使用 1/2 | 描述 |
| --- | --- |
| str.lower() 或 str.upper() | 返回字符串的副本，全部字符小写/大写<br>"AbCdEfGh".lower() 结果为 "abcdefgh" |
| str.split(sep=None) | 返回一个列表，由str根据sep被分隔的部分组成<br>"A,B,C".split(",") 结果为 ['A','B','C'] |
| str.count(sub) | 返回子串sub在str中出现的次数<br>"an apple a day".count("a") 结果为 4 |
| str.replace(old, new) | 返回字符串str副本，所有old子串被替换为new<br>"python".replace("n","n123.io") 结果为 "python123.io" |

# 字符串处理方法

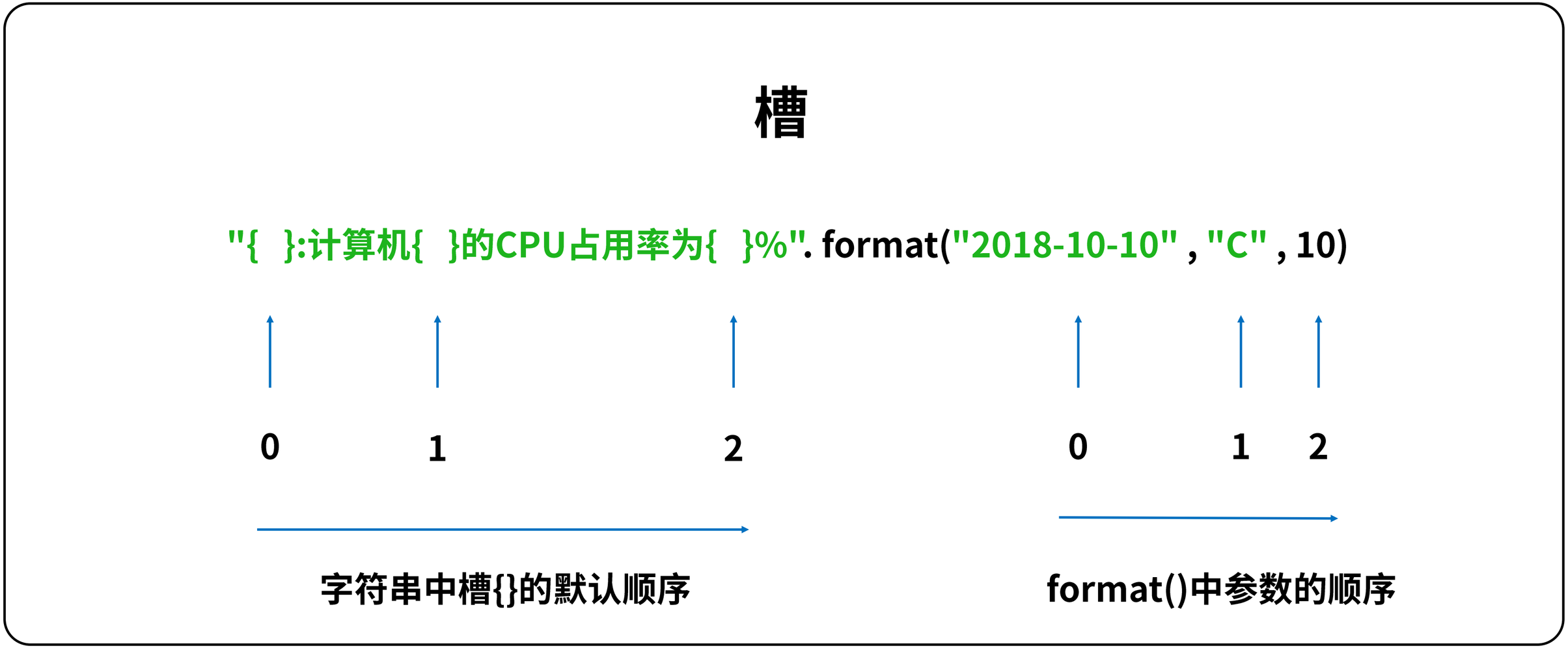
| 方法及使用 2/2 | 描述 |
| --- | --- |
| str.center(width[,fillchar]) | 字符串str根据宽度width居中，fillchar可选<br>"python".center(20,"=") 结果为 '=======python=======' |
| str.strip(chars) | 从str中去掉在其左侧和右侧chars中列出的字符<br>"= python= ".strip(" =np") 结果为"ytho" |
| str.join(iter) | 在iter变量除最后元素外每个元素后增加一个str<br>",".join("12345") 结果为 "1,2,3,4,5" #主要用于字符串分隔等 |

# 字符串的格式化方法

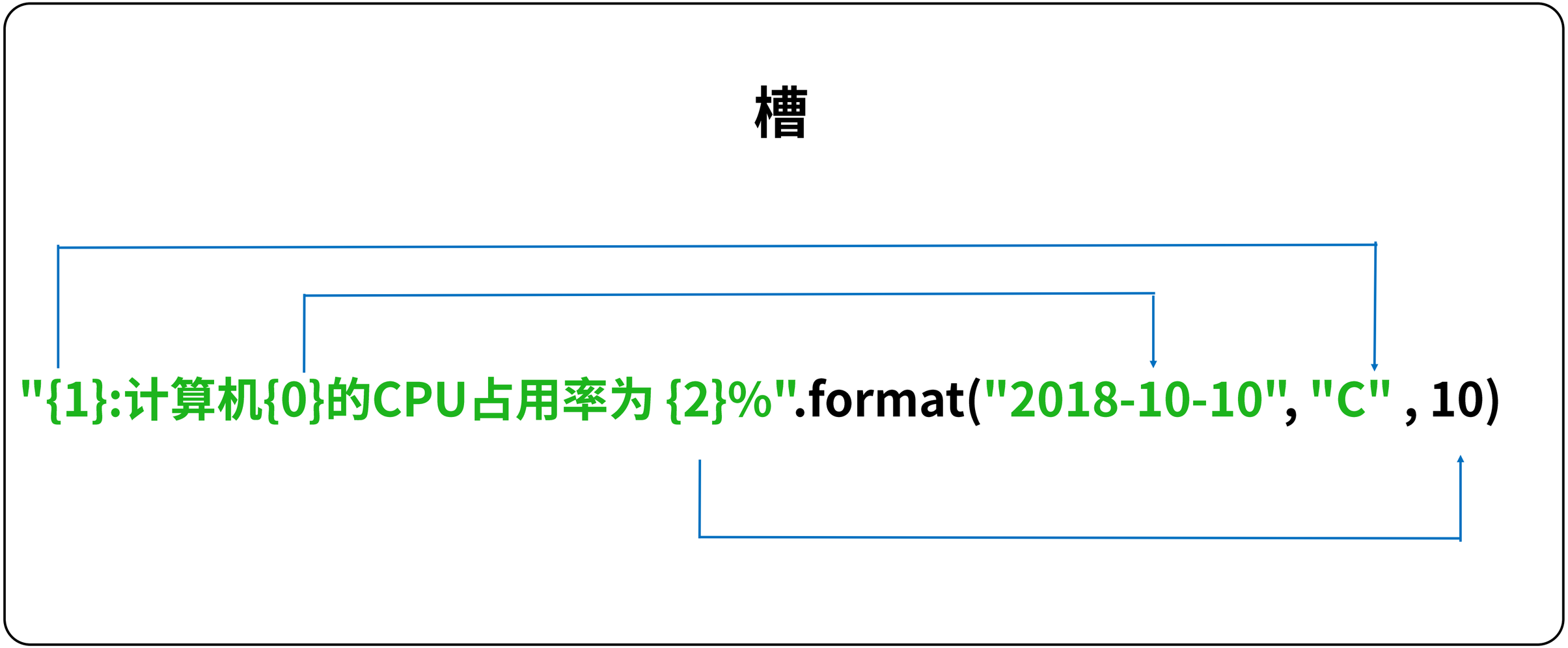
格式化是字符串处理方法中的一种，进行字符串格式表达

<模板字符串>.format(<逗号分隔的参数>)

# 字符串的格式化方法


槽
"{ }:计算机{ }的CPU占用率为{ }%". format("2018-10-10" , "C" , 10)
0
1
2
0
1
2
字符串中槽{}的默认顺序
format()中参数的顺序

# 字符串类型的格式化



# format()槽格式控制

## 槽内部的格式化配置

{ <参数序号>： <格式控制标记>}

| : | <填充> | <对齐> | <宽度> | <,> | <.精度> | <类型> |
|---|---|---|---|---|---|---|
| 引导符号 | 用于填充的单个字符 | < 左对齐<br>> 右对齐<br>^ 居中对齐 | 槽设定的输出宽度 | 数字的千位分隔符 | 浮点数小数精度<br>或 字符串最大输出长度 | 整数类型<br>b, c, d, o, x, X<br>浮点数类型<br>e, E, f, % |

# format()槽格式控制

| : | <填充> | <对齐> | <宽度> | <,> | <.精度> | <类型> |
|---|---|---|---|---|---|---|
| 引导符号 | 用于填充的单个字符 | < 左对齐<br>> 右对齐<br>^ 居中对齐 | 槽设定的输出宽度 | | | |

```
>>>"{0:20}".format("PYTHON")
'PYTHON              '
>>>"{0:=^20}".format("PYTHON")
'=======PYTHON======='
>>>"{0:*>20}".format("PYTHON")
'**************PYTHON'
```

# format()槽格式控制

| : | <填充> | <对齐> | <宽度> | <,> | <.精度> | <类型> |
|---|---|---|---|---|---|---|
| | | | | 数值的千位分隔符 | 浮点数小数精度或 字符串最大输出长度 | 整数类型<br>b, c, d, o, x, X<br>浮点数类型<br>e, E, f, % |

```
>>>"{0:,.2f}".format(12345.6789)
'12,345.68'
>>>"{0:b},{0:c},{0:d},{0:o},{0:x},{0:X}".format(425)
'110101001,Σ,425,651,1a9,1A9'
>>>"{0:e},{0:E},{0:f},{0:%}".format(3.14)
'3.140000e+00,3.140000E+00,3.140000,314.000000%'
```

# 小结：字符串处理方法

## 9个重要的字符串处理方法

- .lower()　　.upper()　　.split()　　.count()
- .replace()　　.center()　　.strip()　　.join()
- .format()

基本数据类型

# 字节串类型

# 字节串

## 字节串：由0个或多个字节组成的有序字节序列

- 单行字节串由一对单引号或一对双引号表示

  b'abcdef 1234567890'

- 多行字节串由三个单引号或三个双引号表示

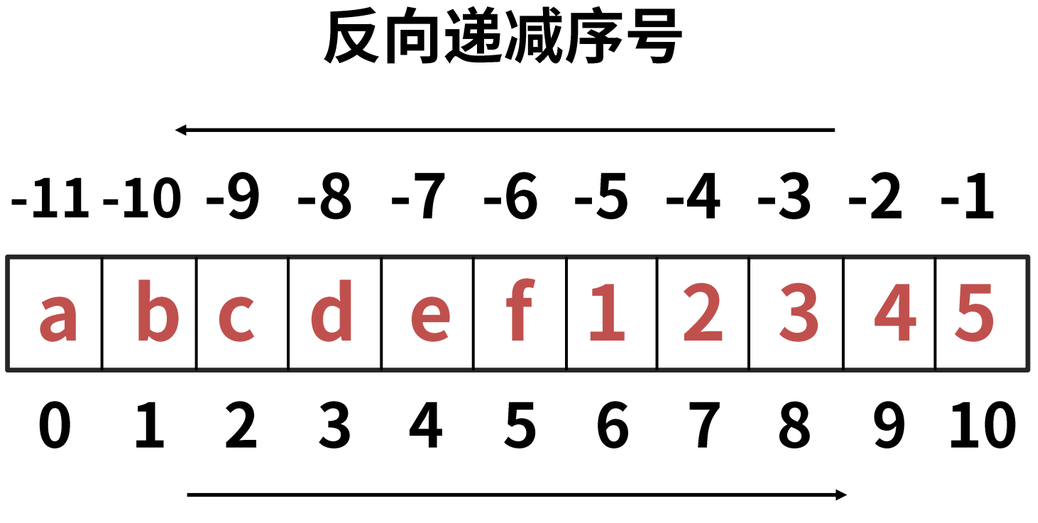
  b'''abcdef

  1234567890 '''

Q: 字节串和字符串的表示方法一样?

字节串中只允许存在ASCII字符

# 字节串的序号



# 字节串的索引和切片

## 使用[ ]获取字节串中一个或多个字节

- 索引：返回字节串中单个字节 <字节串>[M]

  b'abcdef 1234567890'[1] 结果是 "b"

- 切片：返回字节串中一段字节子串 <字节串>[M: N] 不含N

  b'abcdef 1234567890'[1: 5] 结果是 "bcde"

# 字节串切片的高级用法

## 使用[M: N: K]根据步长对字节串切片

- <字节串>[M: N]，M缺失表示至开头，N缺失表示至结尾

  b'abcdef 1234567890'[:3]　　结果是　b"abc"

- <字节串>[M: N: K]，根据步长K对字节串切片

  b'abcdef 1234567890'[1: 8: 2]　结果是 b"bdf1"

  b'abcdef 1234567890'[ ::-1]　　结果是 b"0987654321 fedcba"

# 字节串操作符

| 操作符及使用 | 描述 |
| --- | --- |
| x + y | 连接两个字节串x和y |
| n * x 或 x * n | 复制n次字节串x |
| x in s | 如果x是s的子串，返回True，否则返回False |

基本数据类型

# 字节串操作函数与方法

# 字节串处理函数

| 函数及使用 | 描述 |
| --- | --- |
| len(x) | 长度，返回字节串x的长度 |

# 字节串处理方法

| 方法及使用 1/2 | 描述 |
| --- | --- |
| bytes.lower() 或 bytes.upper() | 返回字节串的副本，全部字节小写/大写<br>b"AbCdEfGh".lower() 结果为 b"abcdefgh" |
| bytes.split(sep=None) | 返回一个列表，由str根据sep被分隔的部分组成<br>b"A,B,C".split(b",") 结果为 [b'A', b'B', b'C'] |
| bytes.count(sub) | 返回子串sub在str中出现的次数<br>b"a apple a day".count(b"a") 结果为 4 |

# 字节串处理方法

| 方法及使用 2/2 | 描述 |
| --- | --- |
| bytes.center(width[,fillbyte]) | 字节串bytes根据宽度width居中，fillbyte可选<br>b"python".center(20, b"=") 结果为 b'=======python=======' |
| bytes.strip(bytes) | 去掉在其左侧和右侧bytes中列出的字符<br>b"= python= ".strip(b" =np") 结果为b"ytho" |
| bytes.replace(old, new) | 返回字节串str副本，所有old子串被替换为new<br>b"python".replace(b"n", b"n123.io") 结果为 b"python123.io" |

基本数据类型

# 单元小结

# 单元小结

**(1) 数值类型**

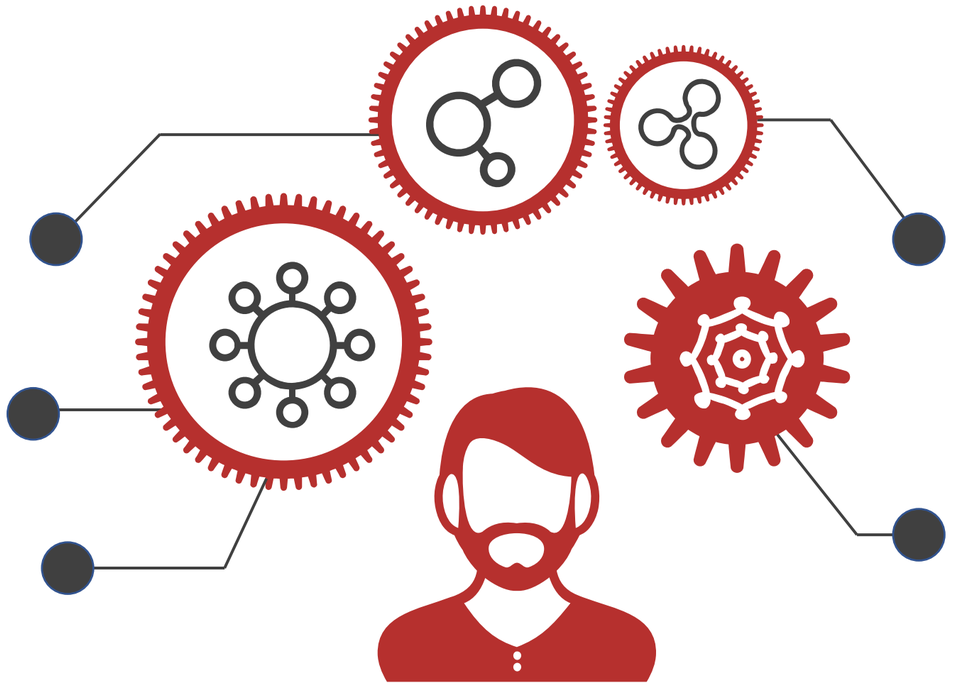
整数、浮点数、复数

**(2) 数值运算函数与方法**

操作符、增强操作符、运算函数

**(3) 整数的位运算**

位运算操作符



**基本数据类型**

**(4) 字符串类型**

索引、切片、转义符、操作符

**(5) 字符串操作函数与方法**

6个函数和9个重要方法

**(6) 字节串类型**

概念、索引、切片、操作符

**(7) 字节串操作函数与方法**

1个函数和7个重要方法


Python▸123
Thank you