安徽理工大学
ANHUI UNIVERSITY OF SCIENCE & TECHNOLOGY

• 人工智能专业 学科基础教育必修模块

Python与机器学习

Python and Machine Learning

Chapter 3：Python Basic Data Types

• Lecturer ：孟亦凡

• E-mail：myf@aust.edu.cn

安徽理工大學
ANHUI UNIVERSITY OF SCIENCE & TECHNOLOGY

Chapter 3： Python Basic Data Types

Target

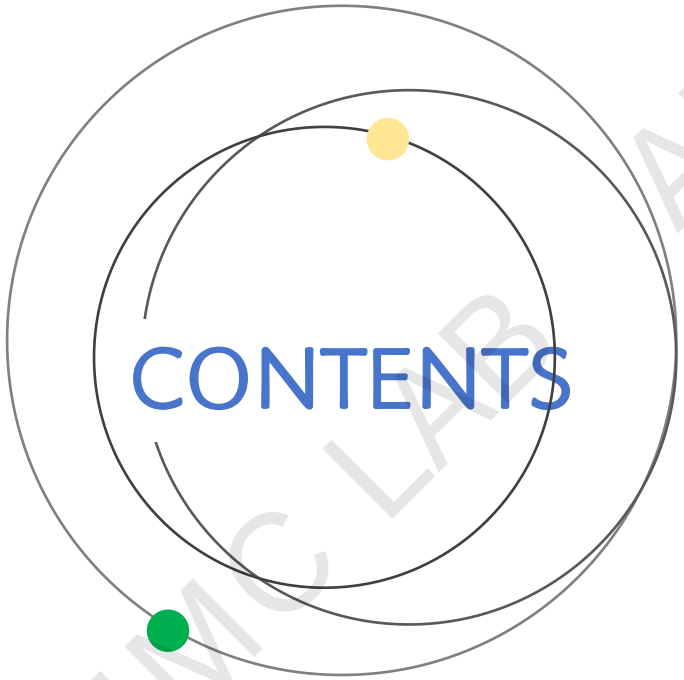
- 理解Python核心数据类型
- **Understand Python's Core Data Types**

- 掌握数字与字符串的操作
- **Master Operations on Numbers and Strings**

- 认识与C++的主要差异
- **Recognize Key Differences from C++**

安徽理工大學
ANHUI UNIVERSITY OF SCIENCE & TECHNOLOGY

Chapter 3: Python Basic Data Types



3.0 Python Basic Data Types

- **The Python language has nine basic data types, which we divide into the following three categories.**

  - **Number Type（数值型）：**

    integer（整型）, float（浮点型）, complex （复数）

  - **Byte Type（字节型）：**

    string（字符串）, byte array（字节串）

  - **Combination Types（组合型）：**

    set（集合）, tuple（元组）, list（列表）, dictionary（字典）

3.1 Basic Data Types & Operations

➢ Integer Type

■ What is Integer?

- **Like the integer meaning in mathematics, it has no range of values;**

- **Integers can be represented in four forms: binary, octal, decimal, and hexadecimal;**

➢ 3.1.1 Integer Type

■ What is Integer?

- **Integers can be represented in four forms: binary, octal, decimal, and hexadecimal;**
  - **Decimal（十进制）： 1010, 99, -217**
  - **Binary（二进制）： 以0b或0B开头： 0b010, -0B101**
  - **Octal（八进制）： 以0o或0O开头： 0o123, -0O456**
  - **Hexadecimal（十六进制）： 以0x或0X开头： 0x9a, -0X89**

➢ 3.1.1 Integer Type

■ Integer variable operation example

- **Integers can be represented in four forms: binary, octal, decimal, and hexadecimal;**

```
num1 = 6
print(num1)
print(type(num1))
```

➢ 3.1.2 Float Type

■ What is Float?

- As with real numbers（实数） in mathematics, numbers with and without decimal places have a range of values（存在取值范围）;
- Floating point numbers include two ways to represent them: conventional method（常规方法） and scientific counting method（科学计数法）;

➢ 3.1.2 Float Type

■ What is Float?

- 科学计数法使用字母e或E作为幂的符号，以10位基数，格式：

  <a>e<b> 表示 $a10^b$；

如：0.0043 = 4.3e-3 ； 98000000.0 = 9.8E7

- 不确定尾数问题（Uncertainty in trailing digits）：浮点数直接运算，可能产生不确定尾数。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。可以使用round()辅助浮点数运算，消除不确定尾数。

3.1 Basic Data Types & Operations

➢ 3.1.2 Float Type

■ Float variable operation example

```
print(0.1 + 0.2)    # 不确定尾数问题
# round(x , b)          #对x四舍五入，d是小数截取位数
print(round(0.1 + 0.2,1)    # 消除不确定尾数
```

```
0.30000000000000004
0.3

进程已结束，退出代码为 0
```

round(x, d): rounds x, d is the number of decimal places to intercept (小数截取位数) .

The round() function is used to help with the comparison between floating point numbers (浮点数间运算) .

Uncertain tails (不确定尾数) generally occur around 10-16, and round() is very effective.

➢ 3.1.3 Complex Number Type

■ What is Complex?

Consistent with the concept of complex numbers in mathematics.

与数学中的复数概念相同，定义 j = $\sqrt{-1}$，复数表示为z = a+bj;

a+bj 被称为复数，其中，a是实部，b是虚部

z.real获得z的实部，z.imag获得z的虚部。

3.1.3 Complex Number Type

Complex Operation Example

```
z = 3 + 4j
print(z)
print(z.real)
print(z.imag)
print(type(z))
```

```
(3+4j)
3.0
4.0
<class 'complex'>

进程已结束，退出代码为 0
```

➢ 3.1.4 Numerical Operators(数值运算操作符)

Operators (操作符) are a system of symbols for completing operations.

操作符及使用	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

➢ 3.1.4 Numerical Operators(数值运算操作符)

Binary operators (二元运算符) have corresponding enhanced assignment operators (增强赋值操作符).

增强操作符及使用	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 x //= y　x %= y　x **= y

	>>> x = 3.1415
	>>> x **= 3　# 与 x = x **3 等价
	31.006276662836743

➢ 3.1.4 Numerical operations functions(数值运算函数)

Some numerical operation functions (数值运算功能) provided in the form of functions (函数形式).

函数及使用	描述
abs(x)	绝对值，x的绝对值 abs(-10.01) 结果为 10.01
divmod(x,y)	商余，(x//y, x%y)，同时输出商和余数 divmod(10, 3) 结果为 (3, 1)
pow(x, y[, z])	幂余，(x**y)%z，[..]表示参数z可省略 pow(3, pow(3, 99), 10000) 结果为 4587

➢ 3.1.4 Numerical operations functions(数值运算函数)

Some numerical operation functions (数值运算功能) provided in the form of functions (函数形式).

函数及使用	描述
round(x[, d])	四舍五入，d是保留小数位数，默认值为0 round(-10.123, 2) 结果为 -10.12
max(x_1,x_2, … ,x_n)	最大值，返回x1 ,x2, … ,xn 中的最大值，n不限 max(1, 9, 5, 4, 3) 结果为 9
min(x_1,x_2, … ,x_n)	最小值，返回x1,x2, … ,xn 中的最小值，n不限 min(1, 9, 5, 4, 3) 结果为 1

➢ 3.1.4 Numerical operations functions(数值运算函数)

Some numerical operation functions (数值运算功能) provided in the form of functions (函数形式).

函数及使用	描述
int(x)	将x变成整数，舍弃小数部分 int(123.45) 结果为123； int("123") 结果为123
float(x)	将x变成浮点数，增加小数部分 float(12) 结果为12.0； float("1.23") 结果为1.23
complex(x)	将x变成复数，增加虚数部分 complex(4) 结果为 4 + 0j

➢ Example:Simplify the four basic operations calculator

```python
# 四则计算器
# Minimal Calculator
# 输入和类型转换
a = float(input("数字1: "))
b = float(input("数字2: "))
op = input("运算符 (+, -, *, /): ")
# 计算
if op == "+":
    result = a + b
elif op == "-":
    result = a - b
elif op == "*":
    result = a * b
elif op == "/":
    result = a / b if b != 0 else "除零错误"
else:
    result = "无效运算符"
# 输出
print(f"结果: {a} {op} {b} = {result}")
print(f"类型: {type(result)}")
```

```
数字1: 12
数字2: 34
运算符 (+, -, *, /): +
结果: 12.0 + 34.0 = 46.0
类型: <class 'float'>

进程已结束，退出代码为 0
```

➢ Summary & Comparison

Integers整数 (int)

- **C++: int, long, short, ...**
- **Python: Only int, automatically handles large numbers（只有int, 自动处理大数）**

Floating Point Numbers浮点数 (float)

- **C++: float, double**
- **Python: Mainly uses float（主要使用float）**

➢ Summary & Comparison

Boolean布尔型 (bool)

- **Python中为int子类 (True=1, False=0)**

Dynamic Typing（动态类型）

```
x = 10    # 自动识别为int
x = 10.5  # 自动转换为float
```

3.2 String Types & Operations

3.2 String Types & Operations

➢ 3.2.1 String Type

An ordered sequence of zero or more characters.

由零个或多个字符组成的有序序列。

- **A string is represented by a pair of single or double quotation marks (单双引号).**

```
str = "请输入带有符号的温度值:"
```

- **A string is an ordered sequence (有序序列) of characters that can be indexed (索引).**

"请" 是 "请输入带有符号的温度值:" 的第0个字符

3.2 String Types & Operations

➢ 3.2.1 String Type

```
>>>'Let's go!'
SyntaxError:invalid syntax
>>>"Let's go!"
"Let's go!"
>>>'Let\'s go!'
"Let's go!"
```

```
>>>'"Hello,Python!" he said'
'"Hello,Python!" he said'
>>>"\"Hello,Python\" he said"
'"Hello,Python!" he said'
```

可以使用反斜杠（\）对引号进行转义

3.2 String Types & Operations

➢ 3.2.1 String Type

■ Concatenation of string

```
>>>x = “Hello,”
>>>y = “python!”
>>>xy
SyntaxError:invalid syntax
```

```
>>>x = “Hello,”
>>>y = “python!”
>>>x+y
‘Hello,python!’
```

3.2 String Types & Operations

➢ 3.2.1 String Type

■ Index of string

正向递增序号 和 反向递减序号

3.2 String Types & Operations

➢ 3.2.1 String Type

■ Use of String

Use [ ] to get one or more characters in a string.

Index: return a single character in the string. **<string>[M]**

TempStr="请输入带有符号的温度值: " **TempStr[0]**

Slice: returns a substring of characters in a string. **<字符串>[M: N]**

TempStr[0:3]

3.2 String Types & Operations

➢ 3.2.1 String Type

Use [M: N: K] to slice the string (字符串切片) according to the step size (步长).

- **<String>[M: N], missing M means to the beginning, missing N means to the end.**

"〇一二三四五六七八九十"[:3] 结果是 "〇一二"

- **<String>[M: N: K], slicing the string according to step K.**

"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"

3.2 String Types & Operations

➢ 3.2.2 String Operation (字符串操作)

An ordered sequence of zero or more characters.

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

3.2 String Types & Operations

➢ 3.2.2 String Operation (字符串操作)

```
>>>“#” * 10
‘##########’
>>>S = “python语言” + “程序设计”
>>>S
‘python语言程序设计’
>>>“python” in S
True
```

➢ 3.2.2 String Operation (字符串操作)

■ String Operation Example

```
#获取星期字符串
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):")) # 输入7
pos=(weekId-1)*3      # 18
print(weekstr[pos: pos+3])   # weekstr[18:21]
```

```
/Users/mengyifan/Document
请输入星期数字(1-7):7
星期日

进程已结束，退出代码为 0
```

3.2 String Types & Operations

➢ 3.2.3 String processing functions (字符串处理函数)

■ String processing functions

Some string handling functions (字符串处理功能) provided in the form of functions (函数形式) .

Function	Description
len(x)	长度，返回字符串x的长度 len("一二三456") 结果为 6
str(x)	任意类型x所对应的字符串形式 str(1.23)结果为"1.23" str([1,2])结果为"[1,2]"
hex(x) 或 oct(x)	整数x的十六进制或八进制小写形式字符串 hex(425)结果为"0x1a9" oct(425)结果为"0o651"

➢ 3.2.3 String processing functions（字符串处理函数）

■ String processing functions

Some string handling functions (字符串处理功能) provided in the form of functions (函数形式) .

Function	Description
chr(u)	u为Unicode编码，返回其对应的字符
ord(x)	x为字符，返回其对应的Unicode编码

➢ 3.2.3 String processing functions（字符串处理函数）

■ Unicode encoding

Unicode assigns a unique numeric code (分配唯一数字编码) to all characters in the world.

- **Unified character encoding, i.e., an encoding that covers almost all characters.**
- **From 0 to 1114111 (0x10FFFF) space, each encoding corresponds to one character。**
- **Every character in a Python string is a Unicode character.**

➢ 3.2.3 String processing functions（字符串处理函数）

■ Unicode encoding

Some interesting examples.

```
>>> "1 + 1 = 2 " + chr(10004)
```

```
>>> "这个字符ㄚ的Unicode值是： " + str(ord("ㄚ"))
```

```
>>> for i in range(12):
        print(chr(9800 + i), end="")
```

➢ 3.2.4 String processing methods(字符串处理方法)

"Method" is a proper term in programming.

- **“method” (方法) refers specifically to the function <b>() in the <a>.<b>() style.**
- **The method itself is also a function (方法本身也是函数) , but related to <a>.**
- **The string or string variable is <a> and there are a number of available methods.**

➢ 3.2.4 String processing methods(字符串处理方法)

Some string handling functions (字符串处理功能) provided in the form of methods (以方法形式) .

```
S1 = "Hello, PYTHON"
S1.lower()
```

```
'hello, python'
```

```
S1 = "Hello, PYTHON"
S1.upper()
```

```
'HELLO, PYTHON'
```

str.lower() 或 str.upper()

返回字符串的副本，全部字符小写/大写

➢ 3.2.4 String processing methods(字符串处理方法)

```
S2 = "A,B,C,D"
S2.split(",")
```

```
['A', 'B', 'C', 'D']
```

```
S2 = "A B C D"
S2.split()
```

```
['A', 'B', 'C', 'D']
```

str.split(sep=None)

返回一个列表，由str根据sep被分隔的部分组成

➢ 3.2.4 String processing methods(字符串处理方法)

```
S3 = "an apple a day"
S3.count("a")
```

```
4
```

str.count(sub)

返回字串sub在str中出现的次数

```
S4 = "python"
S4.replace("n", "n123")
```

```
'python123'
```

str.replace(old,new)

返回字符串str副本，所有old字串被替换成new

➢ 3.2.4 String processing methods(字符串处理方法)

```
S5 = " python "
S5.strip()
```

```
'python'
```

```
S5 = "=python="
S5.strip("=")
```

```
'python'
```

str.strip(chars)

从str中删除首尾chars中列出的字符

➢ 3.2.4 String processing methods(字符串处理方法)

```
','.join("hello")
```

```
'h,e,l,l,o'
```

str.join(iter)

在iter变量除最后元素外

每个元素后增加一个str

主要用于字符串分割

```
S6 = "python"
S6.center(20, "=")
```

```
'=======python======='
```

str.center(width[,fillchar])

字符串str根据宽度width居中，

fillchar填充

➢ 3.2.5 Formatting of string types（字符串格式化）

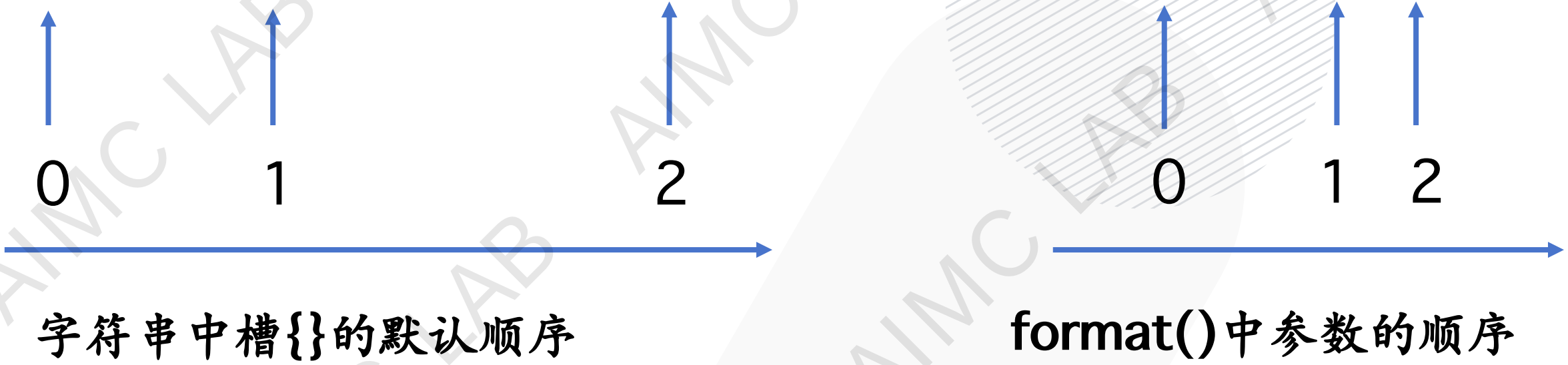
```
TempStr = input("请输入带有符号的温度值:")
if TempStr[-1] in ['F', 'f']:
    C = (eval(TempStr[0:-1]) - 32) / 1.8
    print("转换后的温度是{:.2f}C".format(C))
elif TempStr[-1] in ['C', 'c']:
    F = 1.8*eval(TempStr[0:-1]) + 32
    print("转换后的温度是{:.2f}F".format(F))
else:
    print("输入格式错误！")
```

<模板字符串>.format(<逗号分隔的参数>)

➢ 3.2.5 Formatting of string types（字符串格式化）

槽

```
"{ }:计算机{ }的CPU占用率为{ }%".format("2021-9-27","C",10)
```



➢ 3.2.5 Formatting of string types（字符串格式化）

```
print("{}:计算机{}的CPU占用率为{}%".format("2021-9-27","C",20))
```

```
2021-9-27:计算机C的CPU占用率为20%
```

"{2}:计算机{0}的CPU占用率为{1}%". format("C",20,"2021-9-27")

```
print("{2}:计算机{0}的CPU占用率为{1}%".format("C",20,"2021-9-27"))
```

```
2021-9-27:计算机C的CPU占用率为20%
```

➢ 3.2.5 Formatting of string types（字符串格式化）

槽内部对格式化的配置方式

{ <参数序号> : <格式控制标记>}

```
print("转换后的温度是{0:.2f}C".format(32))
```

```
转换后的温度是32.00C
```

3.2 String Types & Operations

➢ 3.2.5 Formatting of string types（字符串格式化）

参数序号	:	<填充>	<对齐>	<宽度>	<,千位分隔符>	<.精度>	<类型>
	引导符号	用于填充的单个字符	< 左对齐 > 右对齐 ^ 居中对齐	槽设定的输出宽度			

```
"请输出{0:=^20},再输出{1}".format("hello", "python")
```

3.2 String Types & Operations

➢ 3.2.5 Formatting of string types（字符串格式化）

参数序号	:	<填充>	<对齐>	<宽度>	<,千位分隔符>	<.精度>	<类型>
	引导符号	用于填充的单个字符	< 左对齐 > 右对齐 ^ 居中对齐	槽设定的输出宽度			

```
"请输出{0:=^20},再输出{1:*>20}".format("hello", "python")
```

3.2 String Types & Operations

➢ 3.2.5 Formatting of string types（字符串格式化）

```
"请输出{0:10},再输出{1:15}".format("hello", "python")
```

```
'请输出hello     ,再输出python          '
```

```
"{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'
```

3.3 Composite Data Types & Operations

➢ 3.3.1 Set type and operation（集合类型及操作）

■ Definition of set types（集合定义）

Set is an unordered combination of multiple elements.

集合是多个元素的无序组合

- **Set types are consistent with the concept of sets in mathematics.**
  集合类型和数学中的集合概念一致。
- **Unordered between set elements, each element is unique, no identical elements exist.**
  集合元素之间无序，每个元素唯一，不存在相同元素。

➢ 3.3.1 Set type and operation（集合类型及操作）

■ Definition of set types（集合定义）

- **Sets are represented by curly brackets {}, with elements separated by commas.**

  集合用大括号 {} 表示，元素之间用逗号分隔。

- **Create a collection type with {} or set().**

  建立集合类型用 {} 或 set()

- **To create an empty collection type, you must use set().**

  建立空集合类型，必须使用 set()。

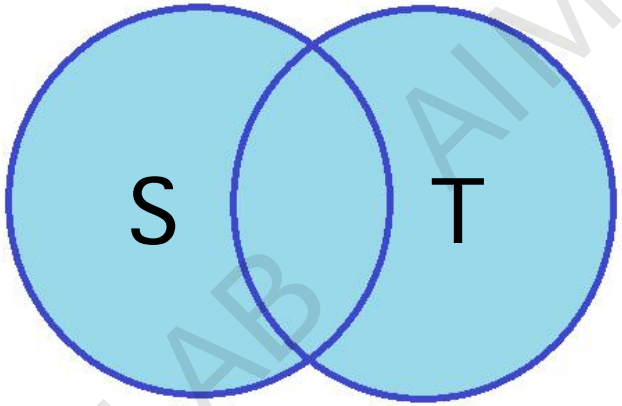
➢ 3.3.1 Set type and operation（集合类型及操作）

■ Definition of set types（集合定义）

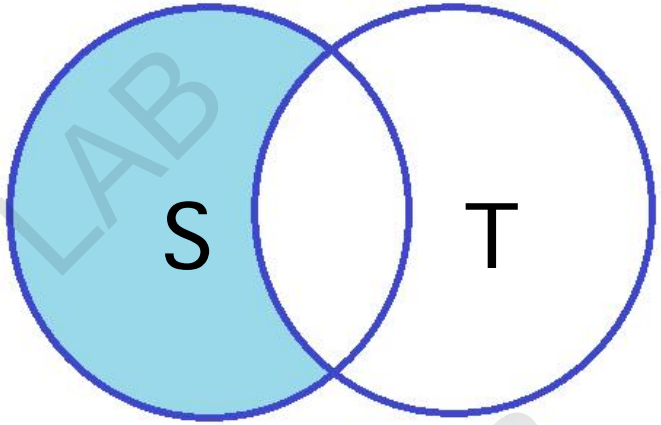
```
>>> A = {"python", 123, ("python",123)}   #使用{}建立集合
{123, 'python', ('python', 123)}
>>> B = set("pypy123")                    #使用set()建立集合
{'1', 'p', '2', '3', 'y'}
>>> C = {"python", 123, "python",123}
{'python', 123}
```

➢ 3.3.1 Set type and operation（集合类型及操作）

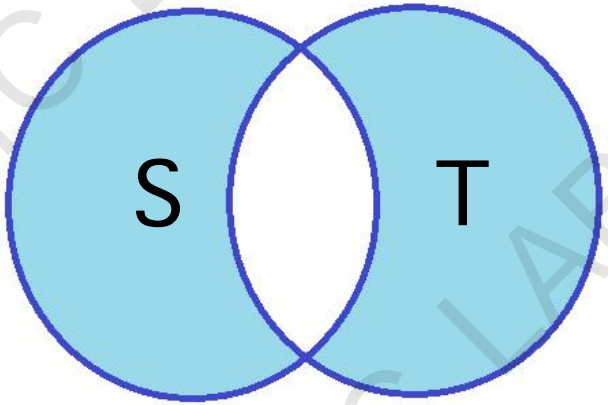
■ Set operator（集合操作符）



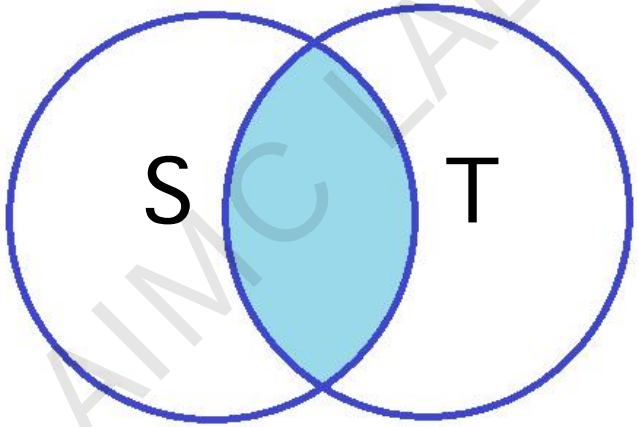
S | T
并



S – T
差



S ^ T
补



S & T
交

➢ 3.3.1 Set type and operation（集合类型及操作）

■ Set operator（集合操作符）

6个操作符

操作符及应用	描述
S | T	并，返回一个新集合，包括在集合S和T中的所有元素
S - T	差，返回一个新集合，包括在集合S但不在T中的元素
S & T	交，返回一个新集合，包括同时在集合S和T中的元素
S ^ T	补，返回一个新集合，包括集合S和T中的非相同元素
S <= T 或 S < T	返回True/False，判断S和T的子集关系
S >= T 或 S > T	返回True/False，判断S和T的包含关系

➢ 3.3.1 Set type and operation（集合类型及操作）

■ Set operator（集合操作符）

```
s = {123, 456, “hello”}
t = {123, “hello”, 456, “python”, 456}
```

```
s | t
{123, 456, “hello”, “python”}
```

```
s & t
{123, 456, “hello”}
```

```
s - t
set()
```

```
t - s
{“python”}
```

```
t ^ s
{“python”}
```

```
t < s
False
```

➢ 3.3.1 Set type and operation（集合类型及操作）

■ Set operator（集合操作符）

4个增强操作符

操作符及应用	描述
S |= T	并，更新集合S，包括在集合S和T中的所有元素
S -= T	差，更新集合S，包括在集合S但不在T中的元素
S &= T	交，更新集合S，包括同时在集合S和T中的元素
S ^= T	补，更新集合S，包括集合S和T中的非相同元素

3.3.1 Set type and operation（集合类型及操作）

Set operator（集合操作符）

```
>>> A = {"p", "y" , 123}
>>> B = set("pypy123")
```

```
>>> A|B
{'1', 'p', '2', 'y', '3', 123}
```

```
>>> A-B
>>> A
{123}
{"p", "y" , 123}
```

```
>>> A -= B
>>> A
{123}
```

➢ 3.3.1 Set type and operation（集合类型及操作）

■ Set processing method（集合处理方法）

操作函数或方法	描述
S.add(x)	如果x不在集合S中，将x增加到S
S.discard(x)	移除S中元素x，如果x不在集合S中，不报错
S.remove(x)	移除S中元素x，如果x不在集合S中，产生KeyError异常
S.clear()	移除S中所有元素
S.pop()	随机返回S的一个元素，更新S，若S为空产生KeyError异常

➢ 3.3.1 Set type and operation（集合类型及操作）

■ Set processing method（集合处理方法）

操作函数或方法	描述
S.copy()	返回集合S的一个副本
len(S)	返回集合S的元素个数
x in S	判断S中元素x，x在集合S中，返回True，否则返回False
x not in S	判断S中元素x，x不在集合S中，返回True，否则返回False
set(x)	将其他类型变量x转变为集合类型

➢ 3.3.1 Set type and operation（集合类型及操作）

■ Set processing method（集合处理方法）

```
>>> A = {"p", "y", 123}
>>> for item in A:
        print(item, end="")
p123y
>>> A
{'p', 123, 'y'}
```

```
>>> try:
        while True:
            print(A.pop(), end=""))
    except:
        pass
p123y
>>> A
set()
```

➢ 3.3.1 Set type and operation（集合类型及操作）

■ Set type application scenarios（集合类型应用场景）

Comparison of inclusion relationships

包含关系比较

```
>>> "p" in {"p", "y" , 123}
True
>>> {"p", "y"} >= {"p", "y" , 123}
False
```

➢ 3.3.1 Set type and operation（集合类型及操作）

■ Set type application scenarios（集合类型应用场景）

Data de-duplication: no duplication of all elements of the set type

数据去重：集合类型所有元素无重复

```
>>> ls = ["p", "p", "y", "y", 123]
>>> s = set(ls)          # 利用了集合无重复元素的特点
{'p', 'y', 123}
>>> lt = list(s)         # 还可以将集合转换为列表
['p', 'y', 123]
```

➢ 3.3.2 Sequence type and operation（序列类型及操作）

■ Sequence Type Definition

Sequence is a set of elements with sequential relationship

序列是具有先后关系的一组元素

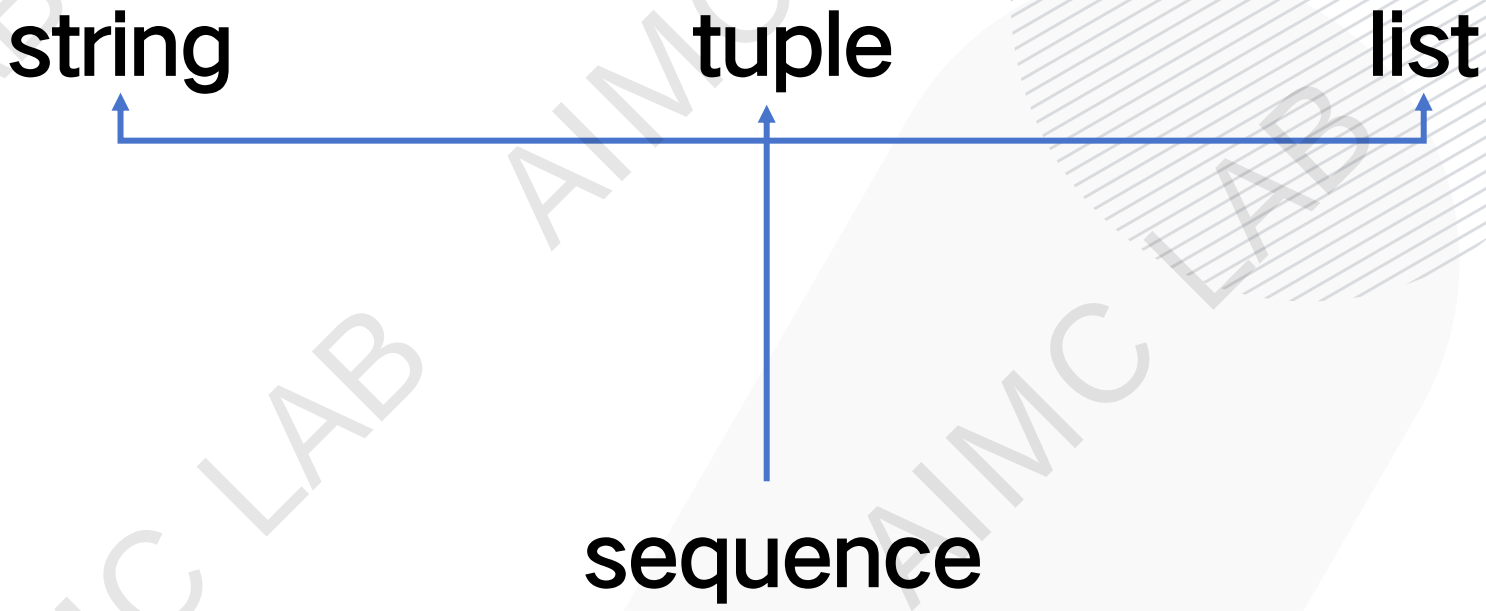
- **Sequences are one-dimensional vectors of elements, and the element types can be different.**
- **Similar sequence of mathematical elements: $S_0, S_1, \ldots, S_{n-1}$**
- **The elements are guided by ordinal numbers, and specific elements of the sequence are accessed by subscripts.**
- [元素间由序号引导，通过下标访问序列的特定元素]

➢ 3.3.2 Sequence type and operation（序列类型及操作）

■ Sequence Type Definition

Sequence is a set of elements with sequential relationship

序列是具有先后关系的一组元素



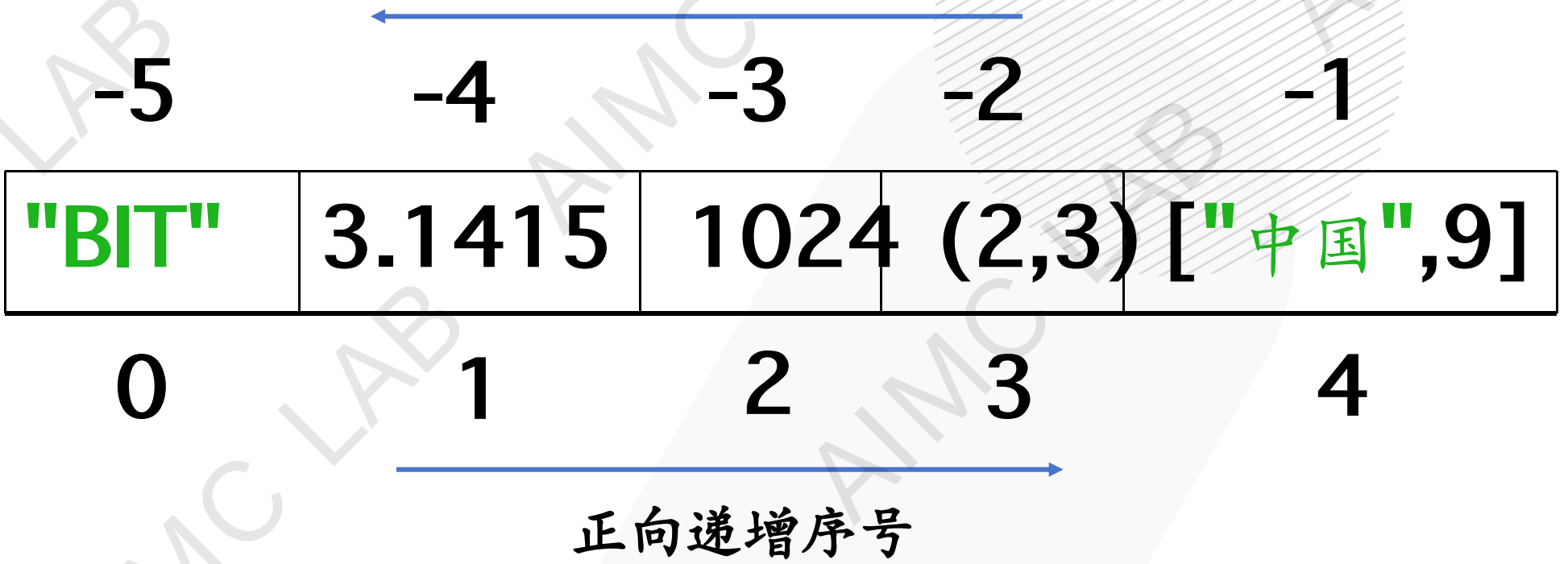
➢ 3.3.2 Sequence type and operation（序列类型及操作）

■ Sequence Type Definition

Definition of serial number

[序号的定义]

反向递减序号



正向递增序号

➢ 3.3.2 Sequence type and operation（序列类型及操作）

■ Sequence processing functions and methods

Sequence type generic operators

[序列类型通用操作符]

操作符及应用	描述
x in s	如果x是序列s的元素，返回True，否则返回False
x not in s	如果x是序列s的元素，返回False，否则返回True
s + t	连接两个序列s和t
s*n 或 n*s	将序列s复制n次
s[i]	索引，返回s中的第i个元素，i是序列的序号
s[i: j] 或 s[i: j: k]	切片，返回序列s中第i到j以k为步长的元素子序列

➢ 3.3.2 Sequence type and operation（序列类型及操作）

■ Sequence type operation example

```
>>> s = "123456"
>>> s[::-1]
'654321'

>>> ls = ["123", 456, "789"]
>>> ls[::-1]
['789', 456, '123']
```

➢ 3.3.2 Sequence type and operation（序列类型及操作）

■ Sequence type operation example

```
>>> s = (2, 3, 4)
>>> t = [2, 3, 4, "Python"]
>>> s in t
    False
>>> s not in t
    True
>>> s + t
    TypeError
>>> 2 * s
    (2, 3, 4, 2, 3, 4)
```

➢ 3.3.2 Sequence type and operation（序列类型及操作）

■ General Functions and Methods for Sequence Types

函数和方法	描述 5个函数和方法
len(s)	返回序列s的长度，即元素个数
min(s)	返回序列s的最小元素，s中元素需要可比较
max(s)	返回序列s的最大元素，s中元素需要可比较
s.index(x) 或 s.index(x, i, j)	返回序列s从i开始到j位置中第一次出现元素x的位置
s.count(x)	返回序列s中出现x的总次数

➢ 3.3.2 Sequence type and operation（序列类型及操作）

■ General Functions and Methods for Sequence Types

```
>>> ls = ["python", 123, ".io"]
>>> len(ls)
3
>>> ls = [12, 35, 8, 20]
>>> max(ls)
35
>>> ls.index(8)
2
```

```
>>> ls = [12, 35, 8, 12]
>>> ls.count(12)
2

>>> ls = [12, 35, “hi”]
>>> max(ls)
TypeError
```

➢ 3.3.2 Tuple type and operation（元组类型及操作）

■ Tuple type definition

Tuple is a special type of sequence type

元组是序列类型中比较特殊的类型

- 元组是一种序列类型，一旦创建就不能被修改
- 使用小括号 () 或 tuple() 创建，元素间用逗号，分隔
- 可以使用或不使用小括号

➢ 3.3.2 Tuple type and operation（元组类型及操作）

■ Tuple type definition

```
>>> c = "cat", "dog", "tiger", "human"
>>> c
('cat', 'dog', 'tiger', 'human')
>>> color = (4352, "blue", c)
>>> color
(4352, 'blue', ('cat', 'dog', 'tiger', 'human'))
```

➢ 3.3.2 Tuple type and operation（元组类型及操作）

■ Tuple type definition

元组继承序列类型的全部通用操作

- 元组继承了序列类型的全部通用操作
- 元组因为创建后不能修改，因此没有特殊操作
- 使用或不使用小括号

➢ 3.3.2 Tuple type and operation（元组类型及操作）

■ Tuple type definition

```
>>> c = "cat", "dog", "tiger", "human"
>>> c[::-1]
('human', 'tiger', 'dog', 'cat')
>>> color = (4352, "blue", c)
>>> color[-1][2]
'tiger'
```

➢ 3.3.2 List type and operation（列表类型及操作）

■ List Type Definition

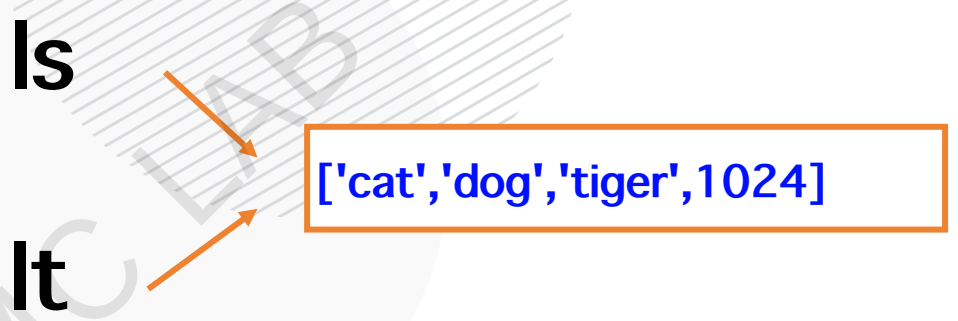
列表是序列类型的一种扩展，十分常用

- 列表是一种序列类型，创建后可以随意被修改
- 使用方括号 [] 或list() 创建，元素间用逗号，分隔
- 列表中各元素类型可以不同，无长度限制

➢ 3.3.2 List type and operation（列表类型及操作）

■ List Type Definition

```
>>> ls = ["cat", "dog", "tiger", 1024]
>>> ls
['cat', 'dog', 'tiger', 1024]
>>> lt = ls
>>> lt
['cat', 'dog', 'tiger', 1024]
```



方括号 [] 真正创建一个列表，赋值仅传递引用

3.3.2 List type and operation（列表类型及操作）

List type operations

```
>>> ls = ["cat", "dog", "tiger", 1024]
>>> ls
['cat', 'dog', 'tiger', 1024]
>>> lt = ls
>>> lt
['cat', 'dog', 'tiger', 1024]
```

方括号 [] 真正创建一个列表，赋值仅传递引用

➢ 3.3.2 List type and operation（列表类型及操作）

■ List type operations

函数或方法	描述
ls[i] = x	替换列表ls第i元素为x
ls[i: j: k] = lt	用列表lt替换ls切片后所对应元素子列表
del ls[i]	删除列表ls中第i元素
del ls[i: j: k]	删除列表ls中第i到第j以k为步长的元素
ls += lt	更新列表ls，将列表lt元素增加到列表ls中
ls *= n	更新列表ls，其元素重复n次

➢ 3.3.2 List type and operation（列表类型及操作）

■ List type operations

```
>>> ls = ["cat", "dog", 10, 24, 1024]
>>> ls[2] = "tiger"
>>> ls
['cat', 'dog', 'tiger', 24, 1024]
>>> ls[::2] = [1, 1, 1]
>>> ls
[1, 'dog', 1, 24, 1]
>>> lt = [123, 456]
>>> lt + ls
[123, 456, 1, 'dog', 1, 24, 1]
```

➢ 3.3.2 List type and operation（列表类型及操作）

■ List type operations

```
>>> ls = ["cat", "dog", "tiger", 1024]
>>> ls[1:2] = [1, 2, 3, 4]
>>> ls
['cat', 1, 2, 3, 4, 'tiger', 1024]
>>> del ls[::3]
[1, 2, 4, 'tiger']
>>> ls*2
[1, 2, 4, 'tiger', 1, 2, 4, 'tiger']
```

```
>>> ls = [1,2,3,4]
>>> ls[1:2] == ls[1]
False
>>> ls[1] = [1,2,3]
>>> ls
[1,[1,2,3],3,4]
```

➢ 3.3.2 List type and operation（列表类型及操作）

■ List type operations functions and methods

函数或方法	描述
ls.append(x)	在列表ls最后增加一个元素x
ls.clear()	删除列表ls中所有元素
ls.copy()	生成一个新列表，赋值ls中所有元素
ls.insert(i,x)	在列表ls的第i位置增加元素x
ls.pop(i)	将列表ls中第i位置元素取出并删除该元素
ls.remove(x)	将列表ls中出现的第一个元素x删除
ls.reverse()	将列表ls中的元素反转

➢ 3.3.2 List type and operation（列表类型及操作）

■ List type operations functions and methods

```
>>> ls = ["cat", "dog", "tiger", 1024]
>>> ls.append(1234)
['cat', 'dog', 'tiger', 1024, 1234]
>>> ls.insert(3, "human")
['cat', 'dog', 'tiger', 'human', 1024, 1234]
>>> ls.reverse()
[1234, 1024, 'human', 'tiger', 'dog', 'cat']
```

➢ 3.3.3 Sequence type application scenarios（序列类型应用场景）

数据表示：元组 和 列表

- 元组用于元素不改变的应用场景，更多用于固定搭配场景
- 列表更加灵活，它是最常用的序列类型
- 最主要作用：表示一组有序数据，进而操作它们

➢ 3.3.3 Sequence type application scenarios（序列类型应用场景）

元素遍历

```
for item in ls :
    <语句块>
```

```
for item in tp :
    <语句块>
```

➢ 3.3.3 Sequence type application scenarios（序列类型应用场景）

数据保护

- 如果不希望数据被程序所改变，转换成元组类型

```
>>> ls = ["cat", "dog", "tiger", 1024]
>>> lt = tuple(ls)
>>> lt
('cat', 'dog', 'tiger', 1024)
```

➢ 3.3.4 Dictionary Type （字典类型）

■ Dictionary Type Definition

理解“映射”

- 映射是一种键(索引)和值(数据)的对应

➢ 3.3.4 Dictionary Type （字典类型）

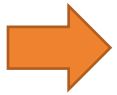
■ Dictionary Type Definition

理解“映射”

- 映射是一种键(索引)和值(数据)的对应

内部颜色：蓝色

外部颜色：红色



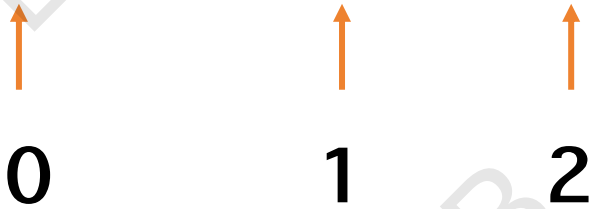
```
"name" : "zhangsan"
"age"  : 18
"ID"   : "2022001"
```

➢ 3.3.4 Dictionary Type （字典类型）

■ Dictionary Type Definition

理解“映射”

- 映射是一种键(索引)和值(数据)的对应



内部颜色：蓝色

外部颜色：红色

序列类型由0..N整数作为数据的默认索引　映射类型则由用户为数据定义索引

➢ 3.3.4 Dictionary Type（字典类型）

■ Dictionary Type Definition

字典类型是“映射”的体现

- 键值对：键是数据索引的扩展

Python3.6之前无序

- 字典是键值对的集合，键值对之间有序
- 采用大括号{}和dict()创建，键值对用冒号:表示

{<键1>:<值1>, <键2>:<值2>, … , <键n>:<值n>}

➢ 3.3.4 Dictionary Type（字典类型）

■ Dictionary Type Definition

在字典变量中，通过键获得值

<字典变量> = {<键1>:<值1>, … , <键n>:<值n>}

<值> = <字典变量>[<键>]　　<字典变量>[<键>] = <值>

[ ] 用来向字典变量中索引或修改或增加元素

➢ 3.3.4 Dictionary Type（字典类型）

■ Dictionary Type Definition

```
>>> d = {"中国":"北京","美国":"华盛顿","法国":"巴黎"}
>>> d
{'中国': '北京', '美国': '华盛顿', '法国': '巴黎'}
>>> d["中国"]
'北京'
>>> d["英国"] = "伦敦"
>>> d
```

{‘中国’: ‘北京’, ‘美国’: ‘华盛顿’, ‘法国’: ‘巴黎’, ‘英国': ‘伦敦’}

➢ 3.3.4 Dictionary Type（字典类型）

■ Dictionary processing functions and methods（字典处理函数及方法）

函数或方法	描述
del d[k]	删除字典d中键k对应的数据值
k in d	判断键k是否在字典d中，如果在返回True，否则False
d.keys()	返回字典d中所有的键信息
d.values()	返回字典d中所有的值信息
d.items()	返回字典d中所有的键值对信息

➢ 3.3.4 Dictionary Type（字典类型）

- Dictionary processing functions and methods（字典处理函数及方法）

```
>>> d = {"name": "zhangsan", "age": 18, "id": "2019001"}
>>> del d['age']
>>> d
{'name': 'zhangsan', 'id': '2019001'}
>>> 'name' in d
True
>>> d.keys()
dict_keys(['name', 'id'])
>>> d.values()
dict_values(['zhangsan', '2019001'])
>>> d.items()
dict_items([('name','zhangsan'),
('id','2019001')])
```

➢ 3.3.4 Dictionary Type（字典类型）

■ Dictionary processing functions and methods（字典处理函数及方法）

函数或方法	描述
d.get(k, <default>)	键k存在，则返回相应值，不在则返回<default>值
d.pop(k, <default>)	键k存在，则取出相应值，不在则返回<default>值
d.popitem()	随机从字典d中取出一个键值对，以元组形式返回
d.clear()	删除所有的键值对
len(d)	返回字典d中元素的个数

3.3.4 Dictionary Type（字典类型）

- Dictionary processing functions and methods（字典处理函数及方法）

```
>>> d = {"中国":"北京","美国":"华盛顿","法国":"巴黎"}
>>> d.get("中国","伊斯兰堡")
'北京'
>>> d.get("巴基斯坦","伊斯兰堡")
'伊斯兰堡'
>>> d.popitem()
('美国', '华盛顿')
>>> d
{"中国":"北京", "法国":"巴黎"}
```

➢ 3.3.4 Dictionary Type（字典类型）

■ Dictionary type application scenarios（字典类型应用场景）

映射的表达

- 映射无处不在，键值对无处不在
- 例如：统计数据出现的次数，数据是键，次数是值
- 最主要作用：表达键值对数据，进而操作它们

3.3.4 Dictionary Type（字典类型）

- Dictionary type application scenarios（字典类型应用场景）

元素遍历

```
for k in d :
    <语句块>
```

```
>>> d = {'a': 1, 'b': 2}
>>> for k in d:
        print(k)
a
b
```

```
>>> d = {'a': 1, 'b': 2}
>>> for k in d:
        print(d.get(k))
1
2
```

3.X Homework

3.X Homework

➢ 学生信息与成绩分析系统

■ **编写一个Python程序，实现学生信息和成绩的综合管理系统，要求综合运用第三章的所有数据类型知识。**

1. 数据输入与验证
- 需要处理的数据类型：
  - 字符串：学生姓名、学号
  - 数字：各科成绩、年龄
  - 列表：多科成绩存储
  - 字典：学生完整信息
  - 集合：课程去重

2. 成绩统计分析
- 统计要求：
  - 计算每个学生的总分、平均分
  - 统计各科最高分、最低分、平均分
  - 分析成绩等级分布

3.X Homework

➢ 学生信息与成绩分析系统

- **编写一个Python程序，实现学生信息和成绩的综合管理系统，要求综合运用第三章的所有数据类型知识。**

3. 数据查询与排序

- 查询功能：
  - 按学号/姓名查询
  - 按总分/单科成绩排序
  - 成绩等级筛选

4. 基础数据结构：使用字典存储学生信息，学号为键

```
students = {
    "2024001": {
        "name": "张三",
        "age": 18,
        "scores": {
            "数学": 85,
            "英语": 92,
            "编程": 78
        }
    }
    # ... 更多学生
}

# 使用集合存储所有课程
courses = {"数学", "英语", "编程"}
```

3.X Homework

➢ 学生信息与成绩分析系统

- 编写一个Python程序，实现学生信息和成绩的综合管理系统，要求综合运用第三章的所有数据类型知识。

```
学生信息与成绩分析系统
综合应用：字符串、数字、列表、字典、集合等数据类型

========================================
1．录入学生信息
2．查询学生信息
3．成绩统计分析
4．按成绩排序
5．生成综合报告
6．退出系统
========================================
请选择功能(1-6)：1
```