Python 基础

Day 03. Python 内置数据类型

1. list 列表

什么是 list

> 可重复 有序 任意类型 可修改 的数据集合

list 的声明

```
names = ['Tom', 'Jerry', 'Spike']
```

何时用 list

> 存储可变可重复的有序数据

常用方法

1. 访问元素

```
names = ['Tom', 'Jerry', 'Spike']

print(names[0])
print(names[-1])
```

2. 列表长度

```
names = ['Tom', 'Jerry', 'Spike']

print(len(names))
```

3. 追加元素

```
names = ['Tom', 'Jerry', 'Spike']

names.append('Tyke')
```

4. 插入元素

```
names = ['Tom', 'Jerry', 'Spike']

names.insert(2, 'Tyke')
```

5. 删除列尾元素

```
names = ['Tom', 'Jerry', 'Spike']
```

```
name = names.pop()

print(name)
```

6. 替换元素

```
names = ['Tom', 'Jerry', 'Spike']

names[2] = 'Tyke'

print(names)
```

7. 清空列表

```
superstars = ['Tom', 'Jerry']
names = [superstars, 'Spike']

names.clear()
```

8. 拷贝

```
superstars = ['Tom', 'Jerry']
names = superstars.copy()

print(names)
print(superstars == names)
```

9. 元素统计

```
```

```
names = ['Tom', 'Jerry']

print(names.count('Tom'))
```

10. 扩展

```
superstars = ['Tom', 'Jerry']
names = ['Spike', 'Tyke']

superstars.extend(names)

print(superstars)
```

11. 返回元素索引

```
names = ['Tom', 'Jerry']

print(names.index('Jerry'))
```

12. 删除第一个指定元素

```
names = ['Tom', 'Jerry', 'Tom']

names.remove('Tom')

print(names)
```

13. 逆序

```
```

```
names = ['Tom', 'Jerry']

names.reverse()

print(names)
```

14. 排序

`list.sort(reverse=True|False, key=myFunc)`

```
names = ['Tom', 'Jerry']

names.sort()
print(names)

names.sort(reverse=True)
print(names)
```

15. 迭代

```
names = ['Tom', 'Jerry']

for name in names:
    print(name)
```

16. 嵌套

```
superstars = ['Tom', 'Jerry']
names = [superstars, 'Spike']
```

```
print(names[0][0])
```

2. tuple 元组 [tjʊpəl; 'tʌpəl]

什么是 tuple

> 可重复 有序 任意类型 不可修改 的数据集合

tuple 的声明

```
names = ('Tom', 'Jerry')

print(names)
```

```
numbers = (1,)

print(numbers)
```

何时用 tuple

> 基于安全的考虑，**能用** tuple 尽量用 tuple

常用方法

1. 访问元素

```
names = ('Tom', 'Jerry')

print(names[-1])
```

2. 判断元素存在

```
names = ('Tom', 'Jerry')

print('Tom' in names)
```

3. 元组长度

```
names = ('Tom', 'Jerry')

print(len(names))
```

4. 元组构造器

```
names = tuple(('Tom', 'Jerry'))

print(names)
```

5. 元素统计

```
names = tuple(('Tom', 'Jerry'))
```

```
print(names.count('Tom'))
```

6. 返回元素索引

```
names = tuple(('Tom', 'Jerry', 'Tom'))

print(names.index('Tom'))
```

7. 不可变的理解

```
superstars = ['Tom']
names = (superstars, 'Spike')

print(names)

names[0].append('Jerry')

print(names)
```

8. 迭代

```
superstars = ['Tom']
names = (superstars, 'Spike')

for name in names:
    print(name)
```

3. dict 字典

什么是 dict

> 字典 `dict` - ionary
>
> 关于 `key - value` 无序的 可修改的 数据集合

dict 的声明

```
d = {'name':'Tom', 'age':18, 'married':False}
```

何时用 dict

> 根据键 **快速** 查找值

- **字典** 的含义
- 原理：dict 根据 key 来计算 value 的存储位置 `hash 算法`
- 类比：list 是顺序查找
- `Space-time tradeoff`

```
print(d['name'])
```

常用方法

1. 声明

```
d = {}
d = dict()
```

2. 初始化

```
d = {'key':'value'}
d = dict(key='value')  #  Constructor
d['key'] = 'value'

# fromkeys 方法
d = {}.fromkeys(['name', 'age'])
print(d)
d = {}.fromkeys(['name', 'age'], 'value')
print(d)
```

3. 获取健对应的值

```
d = {'key': 'value'}
print(d['key'])  # value

# get 方法
print(d.get('key1'))  # None
print(d.get('key1', 'new value'))  # new value
```

4. 更新

```
d = {'key': 'value'}
```

```
d['key'] = 'updated value'
d['new key'] = 'new value'
print(d)
# {'key': 'updated value', 'new key': 'new value'}

# update 方法
d.update({'name': 'Tom'})
print(d)
# {'key': 'updated value', 'new key': 'new value', 'name': 'Tom'}
d.update(age=18)
print(d)
# {'key': 'updated value', 'new key': 'new value', 'name': 'Tom', 'age': 18}
```

5. 删除

```
d = {'name':'Tom', 'age':18, 'married':False}
del d['age']
d.pop('married')
d.popItem()  # removes the last inserted item

print(d)
```

```
d = {'name':'Tom', 'age':18, 'married':False}
del d

print(d)
```

6. 清空

```
d = {'name':'Tom', 'age':18, 'married':False}
```

```
d.clear()

print(d)
```

7. dict 长度

```
d = {'name':'Tom', 'age':18, 'married':False}

print(len(d))
```

8. 拷贝

```
d = {'name':'Tom', 'age':18, 'married':False}

new_dict = d.copy()

print(new_dict)
```

9. 迭代

```
d = {'name': 'Tom', 'age': 18, 'married': False}
print(d.keys())
# dict_keys(['name', 'age', 'married'])

for key in d:
    print(key)

for value in d.values():
    print(value)
```

```
for key, value in d.items():
    print(key, ':', value)
```

dict 注意事项

- dict 的 key 是 **唯一** 的

```
d = {'key':'value'}
print(d['key']) # value

d['key'] = 'new value'
print(d['key']) # new value
```

- 访问不存在的 key 会报错 `KeyError`

```
d = {'key':'value'}
print(d['key'])

#       print(d['key1'])
#   KeyError: 'key1'
```

```
# 判断方法
if 'key1' in d:
    print(d['key1'])

# 或使用 get 方法
```

- dict 的 key 必须是不可变对象

```
list1 = [1, 2, 3]
d1 = {list1: 'value'}
print(d1[list1])

# TypeError: unhashable type: 'list'
```

4. set 集合

什么是 set

> 无序的 不重复的 可修改的 的数据集合

set 的声明

```
keys = {'name', 'age', 'married', 'age'}
```

```
keys = {'name', 'age', 'married', 'age', []}

print(keys)
```

何时用 set

>

> 存储不重复元素

常用方法

1. 添加元素

```
keys = {'name', 'age'}

keys.add('married')

print(keys)
```

2. 删除指定元素

```
keys = {'name', 'age', 'married'}

keys.remove('married')
# keys.discard('married')

print(keys)
```

3. 清空 set

```
names = {'Tom', 'Jerry'}

names.clear()

print(names)
```

4. 拷贝 set

```
names = {'Tom', 'Jerry'}

new_names = names.copy()

print(new_names)
```

5. 随机删除元素

```
names = {'Tom', 'Jerry'}

name = names.pop()

print(name)

print(names)
```

6. 返回差集

```
names1 = {'Tom', 'Jerry'}
names2 = {'Jerry', 'Spike'}

names3 = names1.difference(names2)

print(names3)
```

7. 更新为差集

```
names1 = {'Tom', 'Jerry'}
names2 = {'Jerry', 'Spike'}

names1.difference_update(names2)

print(names1)
```

8. 返回交集

```
names1 = {'Tom', 'Jerry'}
names2 = {'Jerry', 'Spike'}

names3 = names1.intersection(names2)

print(names3)
```

9. 更新为交集

```
names1 = {'Tom', 'Jerry'}
names2 = {'Jerry', 'Spike'}

names1.intersection_update(names2)

print(names1)
```

10. 是否 **不存在** 交集

```
names1 = {'Tom', 'Jerry'}
names2 = {'Jerry', 'Spike'}
```

```
print(names1.isdisjoint(names2))
```

11. 是否为子集

```
names1 = {'Tom', 'Jerry'}
names2 = {'Jerry', 'Spike'}

print(names1.issubset(names2))
```

12. 是否为超集

```
names1 = {'Tom', 'Jerry'}
names2 = {'Jerry', 'Spike'}

print(names1.issuperset(names2))
```

13. 返回对称差集 `[sɪ'metrɪk]`

```
names1 = {'Tom', 'Jerry'}
names2 = {'Jerry', 'Spike'}

names3 = names1.symmectric_difference(names2)

print(names3)
```

14. 更新为对称差集

```
names1 = {'Tom', 'Jerry'}
names2 = {'Jerry', 'Spike'}

names1.symmectric_difference_update(names2)

print(names1)
```

15. 返回并集

```
names1 = {'Tom', 'Jerry'}
names2 = {'Jerry', 'Spike'}

names3 = names1.union(names2)

print(names3)
```

16. 更新为并集

```
names1 = {'Tom', 'Jerry'}
names2 = {'Jerry', 'Spike'}

names1.update(names2)

print(names1)
```

17. 集合运算

```
keys1 = {'name', 'age'}
keys2 = {'age', 'married'}
```

```
print(keys1 & keys2)
print(keys1 | keys2)
```

18. 迭代

```
keys = {'name', 'age', 'married'}

for key in keys:
    print(key)
```

5. 作业

1. 求一个数值 list 的所有元素和
2. 求一个数值 list 的最大/最小元素值
3. 求一个字符串 list 中，字符串长度大于2，且首尾字符相同的元素个数

   > [‘a’, ‘xy’, ‘alabama’, ‘101’]
   >
   > 2

4. 把一个 tuple 转为字符串
5. 对一个 tuple 进行各种切片操作
6. 把 3 个 dict 合并为 1 个 dict
7. 把一个 dict 按 key 排序输出
8. 找出两个 dict 中的 key-value 相同项

   > {‘key1’: 1, ‘key2’: 3, ‘key3’: 2}, {‘key1’: 1, ‘key2’: 2}

```
key1: 1
```

9. 求一个 set 的最大/最小元素值
10. 把两个 set 中的不同元素构造为一个新的 set 并输出