Python速通

1.变量

1.1 命名

常量：CONSTANT_NAME = value
变量：variableName = value

1.2 命名规则

名称	JavaScript	Python
变量名	userName	user_name
类名	UserPermission	UserPermission
函数名	getById	get_by_id
常量名	DEFAULT_LOGO_URL	DEFAULT_LOGO_URL

内置函数	作用
str	将对象转换为字符串
dict	创建字典对象
list	创建列表对象
set	创建集合对象
tuple	创建元组对象
int	将对象转换为整数
float	将对象转换为浮点数
bool	将对象转换为布尔值
len	返回对象的长度
print	将对象打印到控制台

2.基本数据类型

Python 基本数据类型包括：

整数：没有小数点的数字。

浮点数：带有小数点的数字。

字符串：用单引号或双引号括起来的字符序列。

布尔值：表示 True 或 False。

空值：None。

2.1 字符串

```
my_str="FLEETING_SOUNd_123_!"
my_str2="JakeZhang"
print(len(my_str)) #20
print(my_str[::-1]) #!_321_DNUOS_GNITEELF
print(my_str+my_str2) #FLEETING_SOUND_123_!JakeZhang
print(my_str[0:3]) #FLE  substring
print(my_str) #won't change the original string
print(my_str.index(("E"))) #2     stop when finding the first exact char
print(my_str.replace("EE","@@")) #FL@@TING_SOUND_123_!
print(my_str) #won't change the original string
print(my_str.split("E")) #['FL', '', 'TING_SOUND_123_!']
print(my_str.split("N")) #['FLEETI', 'G_SOU', 'D_123_!']
print(my_str.split(" ")) #['FLEETING_SOUNd_123_!']
print(my_str.upper())#FLEETING_SOUND_123_!
print(my_str.lower())#fleeting_sound_123_!

char_list = [char for char in my_str]
print(char_list) #['F', 'L', 'E', 'E', 'T', 'I', 'N', 'G', '_', 'S', 'O', 'U', 'N', 'd', '_',
'1', '2', '3', '_', '!']

print(list(my_str))#['F', 'L', 'E', 'E', 'T', 'I', 'N', 'G', '_', 'S', 'O', 'U', 'N', 'd',
'_', '1', '2', '3', '_', '!']

print(my_str.count("E")) #2
print(my_str.count("fle")) #0   exact match
print(my_str.count("FLE")) #1

print(my_str.startswith("FLE")) #True
print(my_str.endswith("!+"))#  True/False

print("   Hello, World!   ".strip(" ")) #Hello, World!
print("XXXHello, World!YXX".strip("X")) #Hello, World!Y
```

```
"""
1.split("")  不能是空字符串,py不接受空字符串作为分隔符
"""

创建字符串：
str1 = "Hello, World!"

访问字符：
char = str1[0]  # 获取第一个字符

字符串长度：
length = len(str1)

字符串切片：
sub_str = str1[1:5]  # 获取部分字符串

字符串拼接：
str2 = "Python"
concatenated = str1 + " " + str2

字符串重复：
repeated = str1 * 2

字符串遍历：
for char in str1:
    print(char)
检查子串：
exists = "World" in str1  # 返回True或False

字符串分割：
split_str = str1.split(", ")  # 返回分割后的字符串列表

去除空白：
stripped = str1.strip()  # 去除两端空白

替换字符串：
replaced = str1.replace("World", "Python")

转换大小写：
lower_str = str1.lower()
upper_str = str1.upper()
title_str = str1.title()  # 每个单词的首字母大写

查找子串位置：
index = str1.find("World")  # 如果找不到，返回-1

格式化字符串：
formatted = "Hello, {}".format("Python")
formatted_f = f"Hello, {'Python'}"  # f-string
```

```
字符串是否只含字母：
isalpha = str1.isalpha()

字符串是否只含数字：
isdigit = str1.isdigit()

字符串是否只含字母和数字：
isalnum = str1.isalnum()

字符串是否只含空格：
isspace = str1.isspace()

字符串是否以特定子串开头/结尾：
starts_with = str1.startswith("Hello")
ends_with = str1.endswith("World!")

将列表转换为字符串：
list1 = ["Hello", "World"]
joined_str = " ".join(list1)
```

2.2 算术运算

- 在 Python 中，只有 兼容的 数据类型之间才能进行运算，比如浮点数和整数运算的结果为浮点数，而字符串和数字进行运算会抛出异常。如要对数字和字符串进行运算，需先对其中一进行类型转换。
- sum是内置函数名

特性	JavaScript	Python
相加	`+`	`+`
相减	`-`	`-`
相乘	`*`	`*`
相除	`/`	`/`
整除	-	`//`
取余	`%`	`%`
指数运算	`**`	`**`
自增	`++` `+=1`	`+= 1`
自减	`--` `+=1`	`-= 1`

2.3 对象转换

差异速览

特性	JavaScript	Python
转换为整数	`parseInt(myVal)` 或 `Number(myVal)`	`int(my_val)`
转换为浮点数	`parseFloat(myVal)` 或 `Number(myVal)`	`float(my_val)`
转换为字符串	`myVal.toString()` 或 `String(myVal)`	`str(my_val)`
转换为布尔值	`Boolean(myVal)`	`bool(val)`
转换为数组	-	`list(my_val)`
转换为对象	-	`dict(my_val)`

2.4列表和元组(只读的数组)

- 元组可以理解为只读的数组，它在创建时确定元素个数和元素的值，一旦创建就不能被修改。

特性	JavaScript	Python

创建	`let myArr = new Array();`
	`let myArr = [1, 2];`
	`let myTuple = [1, 2];`
		`my_list = list()`
		`my_list = [1, 2]`
		`my_tuple = (1, 2)`

访问	`let el = myArr[index];`	`el = my_list[index]`
添加	`myArr.push(el);`	`my_list.append(el)`
长度	`let length = myArr.length;`	`length = len(my_list)`
切片	`let someEl = myArr.slice(start, end);`	`some_el = my_list[start:end]`
连接	`let mergedArr = myArr1.concat(myArr2);`	`merged_list = my_list1 + my_list2`
复制	`let newArr = [...myArr];`	`new_list = my_list1.copy()`
反转	`myArr.reverse();`	`my_list.reverse()`
删除	`myArr.splice(index, 1);`	`del my_list[index]`
求最大值	`let maxVal = Math.max(...myArr);`	`max_val = max(my_list)`
求最小值	`let minVal = Math.min(...myArr);`	`min_val = min(my_list)`
求和	`let sumVal = myArr.reduce((a,b) => a + b, 0);`	`sum_val = sum(my_list)`
转换为元组	-	`my_tuple = tuple(my_list);`

列表操作

列表操作

创建列表：
```
list1 = [1, 2, 3, 4, 5]
```

访问元素：
```
element = list1[0]  # 访问第一个元素
```

修改元素：
```
list1[0] = 10  # 修改第一个元素
```

添加元素：
```
list1.append(6)  # 在列表末尾添加元素
list1.insert(1, 7)  # 在指定位置插入元素
```

合并列表：
```
list2 = [7, 8, 9]
list1.extend(list2)  # 将list2中的元素添加到list1中
```

删除元素：
```
del list1[0]  # 删除指定位置的元素
list1.remove(5)  # 删除第一个出现的指定元素
list1.pop()  # 删除并返回最后一个元素
list1.pop(0)  #删除并返回第一个元素
```

列表长度：
```
length = len(list1)
```

列表排序：
```
list1.sort()  # 升序排列
list1.sort(reverse=True)  # 降序排列
```

反转列表：
```
list1.reverse()
```

查找元素：
```
index = list1.index(3)  # 返回元素的索引
```

统计元素出现次数：
```
count = list1.count(3)
```

清空列表：
```
list1.clear()
```

列表复制：
```
list2 = list1.copy()
```

使用切片：
```
# 获取部分列表
```

```
sublist = list1[1:3]  # 获取部分列表

列表遍历（普通for循环）：
for item in list1:
    print(item)

列表遍历（带索引）：
for index, item in enumerate(list1):
    print(index, item)

列表推导式（用于创建新列表）：
squared = [x**2 for x in list1]

列表过滤（通过推导式）：
even_numbers = [x for x in list1 if x % 2 == 0]
```

元组操作：

```
创建元组：

tuple1 = (1, 2, 3, 4, 5)
访问元素：

element = tuple1[0]  # 访问第一个元素
元组长度：

length = len(tuple1)
元组切片：

sub_tuple = tuple1[1:3]  # 获取部分元组
元组遍历：

for item in tuple1:
    print(item)
元组拼接：

tuple2 = (6, 7, 8)
concatenated = tuple1 + tuple2
元组重复：

repeated = tuple1 * 2
检查元素是否存在：

exists = 3 in tuple1  # 返回True或False
查找元素索引：

index = tuple1.index(3)
统计元素出现次数：
```

```
count = tuple1.count(3)
```
将列表转换为元组：

```
list1 = [1, 2, 3]
tuple_from_list = tuple(list1)
```
将元组转换为列表：

```
list_from_tuple = list(tuple1)
```
元组解包：

```
a, b, c, d, e = tuple1
```

2.5 字典

- 用get方法访问不一定存在的键，否则可能会引起key error

特性	JavaScript	Python

定义字典	`let obj = {};`
	
	`let obj = {key1: value1, key2: value2};`
		`dict1 = {}`
		
		`my_dict = {"key1": value1, "key2": value2}`

访问值	`obj[key];` key	`my_dict[key]`
访问值或默认值	`obj[key] ?? defaultVal;`	`my_dict.get(key, default_value)`
更新值	`obj[key] = newValue;`	`my_dict[key] = newValue`
合并和更新	`obj={...obj,...anotherObj}`	`my_dict.update(another_dict)`
删除键值对	`delete obj[key];`	`del my_dict[key]`
检查键是否存在	`key in obj;`	`key in my_dict`
获取所有键	`Object.keys(obj);`	`my_dict.keys()`
获取所有值	`Object.values(obj);`	`my_dict.values()`
获取键值对数目	`Object.keys(obj).length;`	`len(my_dict)`

```
personal_info = {}

personal_info["name"] = 'luckrnx09'
personal_info["age"] = 18
personal_info["city"] = '成都'

print(personal_info)

创建字典：
dict1 = {"key1": "value1", "key2": "value2"}
访问元素：
value = dict1["key1"]
修改元素：
dict1["key1"] = "new value"
添加元素：
dict1["key3"] = "value3"
删除元素：
del dict1["key1"]
检查键是否存在：
"key2" in dict1  # 返回True或False
获取所有键：
keys = dict1.keys()
获取所有值：
values = dict1.values()
获取所有键值对：
items = dict1.items()
遍历键：
for key in dict1:
   print(key)
遍历键值对：
for key, value in dict1.items():
   print(key, value)
字典长度：
len(dict1)
字典清空：
dict1.clear()
字典复制：
dict2 = dict1.copy()
从键列表创建字典：
keys = ["a", "b", "c"]
ict3 = dict.fromkeys(keys)

获取字典中键的值，如果键不存在返回默认值：
value = dict1.get("key1", "default value")
删除并返回字典中的最后一个键值对：
item = dict1.popitem()

删除指定键，返回其值：
```

```
删除指定键，返回其值：
value = dict1.pop("key2", "default value")
更新字典：
dict1.update({"key2": "new value2", "key4": "value4"})



node.setdefault('char',{}) 等价于 node=node['char'], if"char" not in node: node["char"]={}
```

2.6 集合set

元素必须是不可变（immutable）类型。这是因为集合本身是一个无序的数据结构，它依赖于其元素的哈希值来维护元素的唯一性和集合的高效性。

1. 数字类型：包括整数（int），浮点数（float），和更复杂的数值类型（如复数complex）。
2. 字符串（string）：字符串是不可变的字符序列，可以被添加到集合中。
3. 元组（tuple）：只要元组中的所有元素都是不可变的（例如，数字、字符串、其它元组），元组本身也可以作为一个元素加入到集合中。

```
text = "apple banana cherry apple banana"
words = text.split(" ")
unique_words = set(words)
sorted_words = sorted(unique_words)
print(sorted_words)
```

特性	JavaScript	Python
创建集合	`let mySet = new Set()`	`my_set = set()`
添加元素	`mySet.add(el)`	`my_set.add(el)`
检查大小	`mySet.size`	`len(my_set)`
检查是否为空	`mySet.size === 0`	`len(my_set) == 0`
删除元素	`mySet.delete(el)`	`my_set.remove(el)`
清空集合	`mySet.clear()`	`my_set.clear()`
检查成员是否存在	`let exist = mySet.has(el)`	`exist = el in my_set`
将集合转换为数组	`let myArr = [...mySet]`	`my_list = list(my_set)`
集合的并集	-	`my_set.union(another_set)`
集合的交集	-	`my_set.intersection(another_set)`

创建集合：

```
set1 = {1, 2, 3, 4, 5}
```

添加元素：

```
set1.add(6)  # 添加单个元素
```

更新集合（添加多个元素）：

```
set1.update([7, 8, 9])  # 添加多个元素
```

集合长度：

```
length = len(set1)
```

删除元素：

```
set1.remove(3)  # 删除指定元素，如果不存在则抛出KeyError
set1.discard(4)  # 删除指定元素，如果不存在不会抛出错误
set1.discard(target_item) #删除指定元素
```

随机删除一个元素并返回：

```
element = set1.pop()
```

清空集合：

```
set1.clear()
```

集合遍历：

```
for item in set1:
```

```
    print(item)

检查元素是否存在：
exists = 5 in set1  # 返回True或False

集合并集：
set2 = {10, 11, 12}
union_set = set1.union(set2)

集合交集：
intersection_set = set1.intersection(set2)

集合差集：
difference_set = set1.difference(set2)

集合对称差集（交集的补集）：
symmetric_difference_set = set1.symmetric_difference(set2)

判断子集：
is_subset = {1, 2}.issubset(set1)

判断超集：
is_superset = set1.issuperset({1, 2})

交集更新：
set1.intersection_update(set2)

差集更新：
set1.difference_update(set2)

对称差集更新：
set1.symmetric_difference_update(set2)

集合复制：
set3 = set1.copy()

冻结集合（创建不可变集合）：
frozenset1 = frozenset([1, 2, 3])
集合主要用于删除重复元素和进行数学上的集合操作，如并集、交集、差集等。由于集合是无序的，所以不能通过索引或切片来访问或修改元素。
```

3.分支和循环

3.1 if语句

3.1 if语句

- `Python 中判断一个值是否为空，可以使用 is None 、 is not None 、== None 和 != None 四种方式。`
- `Python 中没有 switch-case 语句。`

```
number = 100

if number > 50:
   print("50+")
elif number > 0 and number <=50:
   print("<50")
else:
   print("零")
```


特性	JavaScript	Python
if 语句	`if (条件) {代码}`	`if 条件: 代码`
else 语句	`else {代码}`	`else: 代码`
else if 语句	`else if (条件) {代码}`	`elif 条件: 代码`
嵌套 if 语句	相同	相同
逻辑运算符	`&&`、`||`、`!`	`and`、`or`、`not`

是否包含	`key in myObj`
	
	`myArr.includes(el)`
		`key in my_dict`
		
		`el in my_list`

等于	`===`	`==`
不等于	`!==`	`!=`
大于	`>`	`>`
小于	`<`	`<`
大于等于	`>=`	`>=`
小于等于	`<=`	`<=`
条件表达式	`let estimation = score >= 60 ? '及格' : '不及格'`	`estimation = '及格' if score >= 60 else '不及格'`

3.2. for和while语句

差异速览

特性	JavaScript	Python
for 循环	`for (el of iterable)`	`for el in sequence:`
while 循环	`while (condition)`	`while condition:`
range 函数	-	`range(start, stop, step)`
enumerate 函数	-	`enumerate(iterable, start=0)`
break	`break`	`break`
continue	`continue`	`continue`

```
for index,el of enumerate(my_list):
```

4.列表推导和字典推导

在 Python 中，列表推导和字典推导提供了一种简洁的方式来创建列表和字典。

任务

1. 生成一个包含 1 到 10 的平方的数的列表，然后过滤列表中的奇数。
2. 生成一个包含 1 到 10 的平方数的字典，以数字作为键，平方数作为值，过滤字典中值大于等于5的键值对。

```
# 题目 1:
squared_numbers = [x**2 for x in range(1, 11)]
filtered_numbers = [num for num in squared_numbers if num % 2 == 0]
print(filtered_numbers)

# 题目 2:
squared_dict = {x: x**2 for x in range(1, 11)}
filtered_dict = {key: value for key, value in squared_dict.items() if value < 5}
print(filtered_dict)

Python 列表推导用方括号括起来，它返回一个新的列表，
语法是 [表达式 for 项 in 列表 if 条件]。
Python 字典推导使用 {} 括起来，它返回一个新的字典，
语法是 {key表达式: value表达式 for item in 迭代对象 if条件}。

Python 的列表推导和字典推导还可以包括更复杂的条件语句，允许对原始列表/字典进行更复杂的过滤和转化。
```

5.函数的定义和调用

- Python 中，除默认值参数之外（如上面的 `gender` 参数），函数的每个参数都是必传的，否则会出现语法错误。而在 JavaScript 中未传递的参数默认值为 `undefined`。
- 在 Python 中，按照顺序传入的参数叫做 位置参数 ，必须严格按照形参定义顺序传入。此外，还可以使用 `key=value` 的形式向函数传入关键字参数，关键字参数的顺序不必与形参顺序保持一致，位置参数和关键字参数可以一起使用，但 必须保证调用函数时提供了所有必要参数 。
- Python 中，可以使用 `my_fn(*[p1, p2, ...])` 的形式传入位置参数，与 JavaScript 中的 `myFn(...[p1, p2, ...])` 类似。

6.函数入参

使用 *args 获取传入的全部位置参数，类型为元组。使用 **kwargs 获取传入的全部关键字参数，类型为字典。

```
def some_fn(position_arg, *args, **kwargs):
    print(position_arg)
    print(args)
    print(kwargs)

some_fn('position_arg_value','add_to_args1','add_to_args2',a=1,b=2,c=3)
# 运行结果
# position_arg_value
# ('add_to_args1', 'add_to_args2')
# {'a': 1, 'b': 2, 'c': 3}
```

类似地，Python 也可以使用 ** 展开一个字典

```
def some_fn(a, b, c):
    print(a, b, c) # 1 2 3

my_dict = {"c": 3, "b": 2, "a": 1} # Python 对字典中键的顺序没有要求
some_fn(**my_dict)
```

*args 和 **kwargs 虽然方便，但如果函数只定义了这两个形参，IDE 将失去代码提示，同时函数的逻辑也将变得难以理解，需要谨慎使用。

7.Lambda函数

```
lambda params: expression
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda number: number ** 2, numbers))
print(squared_numbers)
#等价于
def square(number): #接收单个参数
    return number ** 2
numbers = [1, 2, 3, 4, 5]
squared_numbers = map(square, numbers) #numbers必须是可迭代对象，例如列表和元组
print(list(squared_numbers))  # 输出: [1, 4, 9, 16, 25]
```

8.创建和导入模块

- Python 模块成员不需要使用 `export` 关键字进行导出，模块中全部成员都将自动导出。
- **Python没有私有函数，除非自定义命名规则，例如_private**
- **引入忽略文件名的大小写**

特性	JavaScript	Python
导入模块	`import foo from 'module'`	`import module`
从模块导入特定成员	`import { foo, bar } from 'module'`	`from module import foo, bar`
将模块导入为别名	`import { foo as alias } from 'module'`	`from module import foo as alias`
将整个模块导入为别名	`import * as alias from 'module'`	`import module as alias`
默认导出	`export default module`	-
命名导出	`export foo`	-

9.创建和导入包

```
import my_package.subpackage # 当不指定模块名称时，默认导入 __init__.py 中的成员，与 JavaScript 项目中的 index.js 文件作用类似
from my_package.subpackage.module3 import my_function # 从 module3.py 中的导入 my_function
```

10.math模块

Python 中的 math 模块提供了各种数学函数和常量。

任务

设计一个程序，通过输入半径来计算圆的面积。

```
import math

radius = 5
area = math.pi * math.pow(radius, 2)
print(area)
```

差异速览

特性	JavaScript	Python
绝对值	`Math.abs(x)`	`abs(x)`
四舍五入到最近的整数	`Math.round(x)`	`round(x)`
向上取整（不小于x的最小整数）	`Math.ceil(x)`	`math.ceil(x)`
向下取整（不大于x的最大整数）	`Math.floor(x)`	`math.floor(x)`
指数运算	`Math.pow(x, y)`	`pow(x, y)`
平方根	`Math.sqrt(x)`	`math.sqrt(x)`
三角函数	`Math.sin(x)`、`Math.cos(x)`、`Math.tan(x)`、`Math.asin(x)`、`Math.acos(x)`、`Math.atan(x)`	`math.sin(x)`、`math.cos(x)`、`math.tan(x)`、`math.asin(x)`、`math.acos(x)`、`math.atan(x)`
将角度转换为弧度	-	`math.radians(x)`
将弧度转换为角度	-	`math.degrees(x)`

11.类型注解

1. 类型注解

编写 Python 代码时，通常会遇到需要明确变量类型的情况。Python 3.5 引入了类型注解，并在后续版本中得到了增强和改进。类型注解不会影响运行时结果，与 TypeScript 十分相似。

使用类型注解
类型注解在 Python 中不是必须的，类型注解是一种在 Python 中指定变量类型的方法，它可以提供更好的代码可读性、可维护性。

```
变量
age: int = 25
name: str = "John"

函数参数和返回值
def add(x: int, y: int) -> int:
    return x + y

可选参数和默认值
from typing import Union
def greet(name: str, age: Union[int, None] = None) -> str:
    if age is None:
        return f"Hello, {name}!"
    else:
        return f"Hello, {name}! You are {age} years old."

类
class Person:
    def __init__(self, name: str, age: int):
        self.name = name
        self.age = age

    def greet(self) -> str:
        return f"Hello, my name is {self.name} and I am {self.age} years old."

类型别名
from typing import List, Tuple

Coordinates = Tuple[float, float]
PointList = List[Coordinates]

def get_distance(point1: Coordinates, point2: Coordinates) -> float:
    x1, y1 = point1
    x2, y2 = point2
    return ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5

def process_points(points: PointList) -> List[float]:
    distances = []
    for i in range(len(points) - 1):
        distance = get_distance(points[i], points[i+1])
```

```
            distances.append(distance)
        return distances
```

编译时类型检查
在 Python 中的 mypy 一个类似于 TypeScript 的 tsc命令行工具，使用它可以对代码的类型进行检查。

```
pip install mypy
mypy my_script.py
```

12.类和对象

```
import math

class Circle:
  pi = math.pi
  def __init__(self, radius):
    self.radius = radius

  @classmethod
  def create_circle(cls,radius):
    return Circle(radius)


  def get_area(self):
    return Circle.pi * self.radius ** 2

  def get_circumference(self):
    return 2 * Circle.pi * self.radius

  def __str__(self):
    return f"[Circle] radius:{self.radius}"

circle = Circle(5) # 或 Circle.create_circle(5)
print(circle) # [Circle] radius:5
print(circle.get_area()) # 78.53981633974483
print(circle.get_circumference()) # 31.41592653589793
```

- Python 像调用函数一样创建类实例，JavaScript 必须使用 `new` 关键字。
- Python 中使用 `__init__` 方法作为构造函数，JavaScript 使用 `constructor` 方法。

- Python 类的实例方法的第一个参数必须为 `self` ，它表示类的实例。类方法需要使用 `classmethod` 装饰器进行

标记，第一个参数必须为 `cls`，表示当前类，也可以使用 `staticmethod` 装饰器，它跟普通函数一样，不需要第一个参数为 `cls`。

- Python 使用缩进来定义类中的代码块，JavaScript 使用大括号 `{}` 表示代码块。
- Python 类中以 `_` 开头的成员约定为私有成员，外部不应该直接使用，在 TypeScript 中可以使用 `private`、`protected` 访问修饰符对可访问性进行限制。
- Python 中以 `__` 开头的为一般为系统级的特殊成员，在外部不可访问。

PYTHON 类中的特殊方法

Python 中有许多常用的特殊方法，开发者可以在类中对它进行重写。

以下为部分常见的特殊方法

- **init**: 类构造函数，用于对象的初始化。当创建类的实例时，方法会被自动调用。
- **str**: 用于返回对象的字符串表示。当使用 `print` 函数打印对象时，会调用该方法。
- **repr**: 用于返回对象的可打印字符串表示。当在交互式环境中直接输入对象名时，会调用该方法。
- **len**: 用于返回对象的长度。当使用 `len` 函数获取对象的长度时，会调用该方法。
- **getitem**: 用于通过索引访问对象的元素。当使用索引运算符（[]）访问对象时，会调用该方法。
- **setitem**: 用于通过索引设置对象的元素。当使用索引运算符（[]）设置对象时，会调用该方法。
- **delitem**: 用于通过索引删除对象的元素。当使用 `del` 关键字删除对象的元素时，会调用该方法。
- **getattr**: 用于在访问不存在的属性时触发。当访问对象不存在的属性时，会调用该方法。
- **setattr**: 用于在设置属性时触发。当设置对象的属性时，会调用该方法。
- **delattr**: 用于在删除属性时触发。当使用 `del` 关键字删除对象的属性时，会调用该方法。

差异速览

特性	JavaScript	Python

定义类	`class ClassName {}`	`class ClassName:`
创建对象	`let myObj = new ClassName();`	`my_obj = ClassName()`
构造函数	`constructor() {}`	`def __init__(self):`
类属性赋值	`this.propertyName = value;`	`self.propertyName = value`
类方法	`methodName() {}`	`def methodName(self):`
继承	`class ChildClass extends ParentClass {}`	`class ChildClass(ParentClass):`
对象销毁	无内置支持	`def __del__(self):`

13.try-except 语句

```
try:
    # 可能会引发异常的代码
    a = 10 / 0  # 除以 0 会引发 ZeroDivisionError
    # 注意：下面的代码不会执行，因为上一行已经引发了异常
    b = int('abc')  # 'abc' 无法转换为整数会引发 ValueError
    raise Exception('This is an exception')  # 主动抛出异常

except ZeroDivisionError:
    # 处理 ZeroDivisionError 异常的代码
    print("除数不能为零！")

except ValueError:
    # 处理 ValueError 异常的代码
    print("无法将字符串转换为整数！")

except Exception as e:
    # 处理其他类型的异常的代码
    print("发生了其他异常:", str(e))
```

错误捕获范围应该从小到大

上面的代码中，最后一个 Exception 类型的异常是所有异常的基类，只有当前面所有的异常类型都不匹配时，它才会执行，这让程序不论发生什么错误都始终能被捕获。

在 Python 代码中，使用 raise 可以将捕获到的异常或自定义的异常进行抛出，与 JavaScript 中的 throw 功能一样。

自定义异常

```
# 定义自定义异常
```

```python
class InvalidShapeException(Exception):
    def __init__(self, message):
        super().__init__(message)

class NegativeDimensionException(Exception):
    def __init__(self, message):
        super().__init__(message)

def calculate_rectangle_area(length, width):
    if not isinstance(length, (int, float)) or not isinstance(width, (int, float)):
        raise InvalidShapeException("无效的尺寸。长度和宽度必须是数字。")
    if length <= 0 or width <= 0:
        raise NegativeDimensionException("无效的尺寸。长度和宽度必须是正数。")
    return length * width

print(calculate_rectangle_area(5, 4)) # 输出：20
print(calculate_rectangle_area("5", 4)) # 引发InvalidShapeException
print(calculate_rectangle_area(-5, 4)) # 引发NegativeDimensionException
```

- Python 异常基类是 `Exception`，而 JavaScript 异常基类是 `Error`，它们都使用 `class` 继承基类实现自定义异常。
- **Python 使用 `isinstance()` 函数来检查变量的类型，而 JavaScript 使用 `typeof` 和 `instance of`。**

特性	JavaScript	Python
创建自定义异常	`class CustomException extends Error {}`	`class CustomException(Exception):`
引发自定义异常	`throw new CustomException(message)`	`raise CustomException(message)`