

第五周

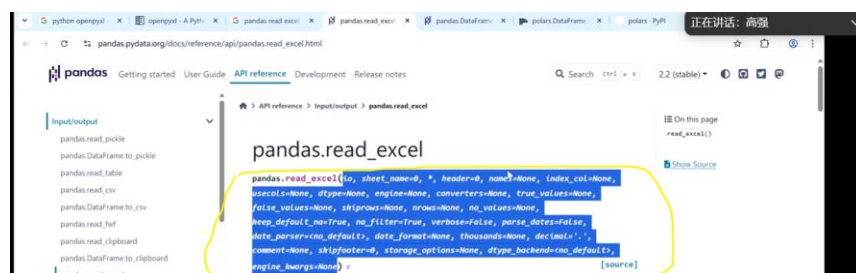
一、编程步骤

(一) 下载 安装包

Pandas(有些过时)里的 xx (pd) 软件包

Polars (很强, 新)

(二) 找软件包需要调用什么函数, 找函数说明书(如下)



看看 example

黄色: 函数签名=函数名称+参数(形参)(蓝色部分)

注意: 只有第一个 io 参数必须给, 其他的可以不给, 有默认值为 0

Io 可以是字符串、字节串等, 最简单的就是给一个“相对路径”(需要处理文件的文件名)

函数最后都有返回值:

Returns:

DataFrame or dict of DataFrames

DataFrame from the passed in Excel file. See notes in sheet_name argument for more information on when a dict of DataFrames is returned.

返回为: DataFrame: 二维的表格型数据

```
main.py > ...  
1 import pandas as pd  
2  
3 df = pd.read_excel("选课名单-24专硕")
```

给函数的返回值, 起个名字叫 df

(三) 设置完 main.py 后, 终端开始调试与调用

```
(base) qiang@gqm3win CLANGARM64 ~/repo/names
$ conda activate names
(names)
qiang@gqm3win CLANGARM64 ~/repo/names
$ ls -l
total 50
-rw-r--r-- 1 qiang 197608 127 3月 31 08:25 environment.yml
-rw-r--r-- 1 qiang 197608 21489 3月 17 08:29 GitCode_平台用户名收集.xlsx
-rw-r--r-- 1 qiang 197608 226 3月 31 08:56 main.py
-rw-r--r-- 1 qiang 197608 4895 3月 24 09:27 sample.xlsx
-rw-r--r-- 1 qiang 197608 13834 3月 17 07:50 选课名单-24专硕.xlsx
(names)
qiang@gqm3win CLANGARM64 ~/repo/names
$ cat main.py
import pandas as pd

df = pd.read_excel("选课名单-24专硕")

from openpyxl import load_workbook

wb = load_workbook(filename="empty_book.xlsx")
sheet_ranges = wb["range names"]
print(sheet_ranges["D18"].value)
```

(四) 开始 pdb 调试:

```
(names)
qiang@gqm3win CLANGARM64 ~/repo/names
$ python -m pdb main.py
> c:\users\qiang\repo\names\main.py(1)<module>()
-> import pandas as pd
(Pdb) l
1  -> import pandas as pd
2
3      df = pd.read_excel("选课名单-24专硕")
4
5
6      from openpyxl import load_workbook
7
8      wb = load_workbook(filename="empty_book.xlsx")
9      sheet_ranges = wb["range names"]
10     print(sheet_ranges["D18"].value)
[EOF]
```

遇到各种报错,

谷歌搜索 pandas DataFrame

attributes 就是它的属性; method 是方法

Attribute:	
T	The transpose of the DataFrame.
at	Access a single value for a row/column label pair.
attrs	Dictionary of global attributes of this dataset.
axes	Return a list representing the axes of the DataFrame.
columns	The column labels of the DataFrame.
dtypes	Return the dtypes in the DataFrame.
empty	Indicator whether Series/DataFrame is empty.
flags	Get the properties associated with this pandas object.

最重要的属性: [columns](#) The column labels of the DataFrame. 用来看有哪些列

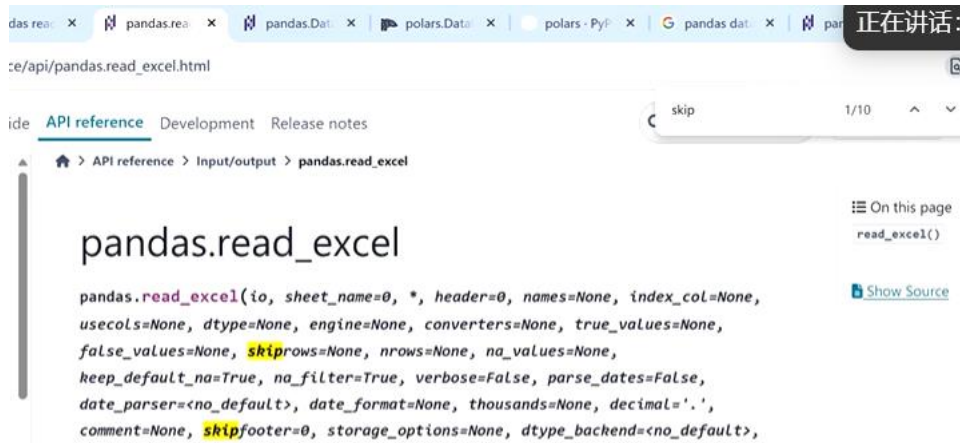
```
(Pdb) p df.columns
Index(['金融编程与计算', 'Unnamed: 1', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4',
      'Unnamed: 5', 'Unnamed: 6', 'Unnamed: 7', 'Unnamed: 8'],
      dtype='object')
```

Methods	
<code>abs()</code>	Return a Series/DataFrame with absolute numeric value of each element.
<code>add(other[, axis, level, fill_value])</code>	Get Addition of dataframe and other, element-wise (binary operator <i>add</i>).

需求 1: 读取 excel 时候, 忽略前三行——做法如下:

谷歌搜索 pandas DataFrame

Ctrl F:搜索



然后用 pdb 调用这个 excel 所在的那行: 要把那一行抄一遍:

并看看 skip 怎么用的, 学会了, 一并输入

```
(Pdb) p pd.read_excel("选课名单-24专硕.xlsx", skiprows=3)
```

成绩	成绩	备注	学号	姓名	院系名称	专业名称	考勤\n成绩	作业\n成绩	期末\n成绩
0	22024110031	马莹超		金融学院	金融	NaN	NaN	NaN	NaN
1	22024110032	马笑璐		金融学院	金融	NaN	NaN	NaN	NaN
2	22024110033	牛文		金融学院	金融	NaN	NaN	NaN	NaN
3	22024110034	马博贤		金融学院	金融	NaN	NaN	NaN	NaN
4	22024110035	蒯一博		金融学院	金融	NaN	NaN	NaN	NaN
...
110	22024110167	王慧敏		金融学院	金融	NaN	NaN	NaN	NaN
111	22024110168	郑涵天		金融学院	金融	NaN	NaN	NaN	NaN
112	22024110169	张涵		金融学院	金融学本硕博	NaN	NaN	NaN	NaN
113	22024110170	祝宝莹		金融学院	金融	NaN	NaN	NaN	NaN
114	22024110171	宋凯迪		金融学院	金融	NaN	NaN	NaN	NaN

成功了, 就把代码更新到 vs code 的 main.py 里

```
main.py > ...
1 import pandas as pd
2
3 df = pd.read_excel("选课名单-24专硕.xlsx", skiprows=3)
4
```

保存 main.py

在终端里 q 退掉 pdb, 因为代码改了, 就要让 pdb 重新运行

需求 2: 只读取 EXCEL 的前四列——做法如下:

谷歌搜索 pandas DataFrame

Ctrl F:搜索

查阅官方这个函数的文档

找到如何改 main.py, 然后改 main.py 代码

```
main.py > ...
1 import pandas as pd
2
3 df = pd.read_excel("选课名单-24专硕.xlsx", skiprows=3, usecols="A:D")
4
```

技巧:

breakpoint()函数

自动调出 pdb, 并且让 pdb 自动运行到 breakpoint()插入的代码位置, 你把 breakpoint()放在代码的哪一行, pdb 就会自动运行到这一行, 然后停在在在一行

——作用: 快速打开 pdb 里我正在调试的断点

举例:

```
! environment.yml  main.py 1 X
main.py > ...
1 import pandas as pd
2
3 df = pd.read_excel("选课名单-24专硕.xlsx", skiprows=3, usecols="A:D")
4 breakpoint() I
5
```

假如我非常确认 df 那一行及以前的所有行代码都被我调好了, 都是是我要的了, 我下面要调试这一行下面的内容, 那就可以把 breakpoint()放在这一行下面, 然后 q 退出 pdb,,直接输入 python main.py 运行 main.py 里的代码即可, 不用打开 pdb,因为 breakpoint()可以做到自动打开 pdb

```
qiang@gqm3win CLANGARM64 ~/repo/names
$ python main.py
> c:\users\qiang\repo\names\main.py(7)<module>()
-> from openpyxl import load_workbook
(Pdb) l
2
3 df = pd.read_excel("选课名单-24专硕.xlsx", skiprows=3,
4 breakpoint()
5
6
7 -> from openpyxl import load_workbook
8
9 wb = load_workbook(filename="empty_book.xlsx")
10 sheet_ranges = wb["range names"]
11 print(sheet_ranges["D18"].value)
```

```
(Pdb) import wat
(Pdb) wat / t
```

下一步就可以: t:我在 vscode 的代码里, 令 t=xxx 了, 我现在就想

```
(Pdb) import wat
(Pdb) wat / t
```

知道 t 是个什么东西以及它的所有属性, 我就 就能查看了。

二、老师的笔记

任务目标

上周 (初级) 我们讲过 Python 编程本质上是拼接操纵各种对象, 因而在本周, 我们的目标是要掌握最基础、最常用的几种 Python 对象类型 (type), 包括字符串 (str)、字节串 (bytes)、整数 (int)、浮点数 (float)、布尔值 (bool)、列表 (list)、字典 (dict)、元组 (tuple)、集合 (set)。这几种类型都是 Python 解释器 内置的 (built-in), 不需要任何导入 (import)。

另外, Python 标准库 (standard library) 里的 pathlib 和 datetime 模块 (module) 提供了用于处理 路径 和 日期时间 的类型, 也是非常基础、非常常用的。标准库模块都不需要安装 (pip/conda install), 但使用前需要导入 (import)。

1. Fork 第 05 周打卡 仓库至你的名下, 然后将你名下的这个仓库 Clone 到你的本地计算机
2. 用 VS Code 打开项目目录, 新建一个 environment.yml 文件, 指定安装 Python 3.12, 然后运行 conda env create 命令创建 Conda 环境
3. 逐个创建 use_of_{name}.py 文件, 其中 {name} 替换为上述要求掌握的对象类型,

例如 `use_of_str.py`:

- 在全局作用域 (global scope) 内尝试键入 (活学活用) Python 代码, 亲手验证概念 (Proof of Concept, PoC)
 - 对于任何对象, 都可以传给以下内置函数 (built-in function) 用于检视 (inspect):
 - `id()` -- 返回对象在虚拟内存中的地址 (正整数), 如果 `id(a) == id(b)`, 那么 `a is b` (`is` 是个运算符)
 - `type()` -- 返回对象的类型
 - `isinstance()` -- 判断对象是否属于某个 (或某些) 类型
 - `dir()` -- 返回对象所支持的属性 (attributes) 的名称列表
 - `str()` -- 返回对象 `print` 时要显示在终端的字符串
 - 可以调用 `print()` 函数将表达式 (expression) 输出到终端, 查看结果是否符合预期
 - 可以利用 `assert` 语句查验某个表达式 (expression) 为真, 否则报错 (`AssertionError`) 退出
 - 可以利用 `try` 语句拦截报错, 避免退出, 将流程 (flow) 转入 `except` 语句
 - 可以调用 `breakpoint()` 函数暂停程序运行, 进入 `pdb` 调试 (debug) 模式
4. 对于 每一个上述要求掌握的对象类型 (将来遇到新的对象类型也应该如此), 我们首先应该熟悉如何通过 **表达式** (expression) 得到他们的 **实例** (instance), 一般包括以下途径:
- 字面值 (literal) (包括 f-string 语法)
 - 推导式 (comprehension) (仅限 list、dict、set)
 - 初始化 (init)
 - 运算值 (operator)
 - 索引值 (subscription)
 - 返回值 (return value of function/method call)
5. 对于 每一个上述要求掌握的对象类型 (将来遇到新的对象类型也应该如此), 我们也要尝试验证其以下几个方面的 **属性** (attributes):
- 对数学运算符 (+、-、*、**、/、//、%、@) 有没有支持
 - 如何判断相等 (==)
 - 对于比较运算符 (>、<、>=、<=) 有没有支持
 - 什么值被当作 True, 什么值被当作 False
 - 是否可迭代 (iterable), 如何做迭代 (for 循环)
 - 是否支持返回长度 (len)
 - 是否 (如何) 支持索引操作 (subscription) (`[]` 运算符)
 - 拥有哪些常用方法 (method) 可供调用 (`()` 运算符)

建议先在 `pdb` 里试验, 然后把确定能够运行的代码写在 `use_of_{name}.py` 文件里

三、Python 对象类型


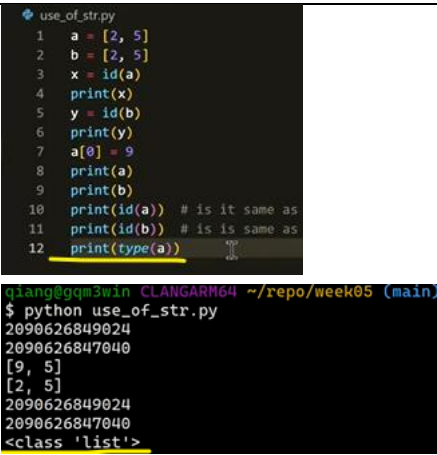
最常用的几种 Python 对象类型 (type), 包括字符串 (str)、字节串 (bytes)、整数 (int)、浮点数 (float)、布尔值 (bool)、列表 (list)、字典 (dict)、元组 (tuple)、集合 (set)。这几种类型都是 Python 解释器 内置的 (built-in), 不需要任何导入 (import)。

另外, Python 标准库 (standard library) 里的 **pathlib** 和 **datetime** 模块 (module) 提供了用于处理 路径 和 日期时间 的类型, 也是非常基础、非常常用的。标准库模块都不需要安装 (pip/conda install), 但使用前需要导入 (import)。

(一) 字符串 str

1. 全局作用域 (global scope) 内尝试键入 (活学活用) Python 代码, 亲手验证概念 (Proof of Concept, PoC)

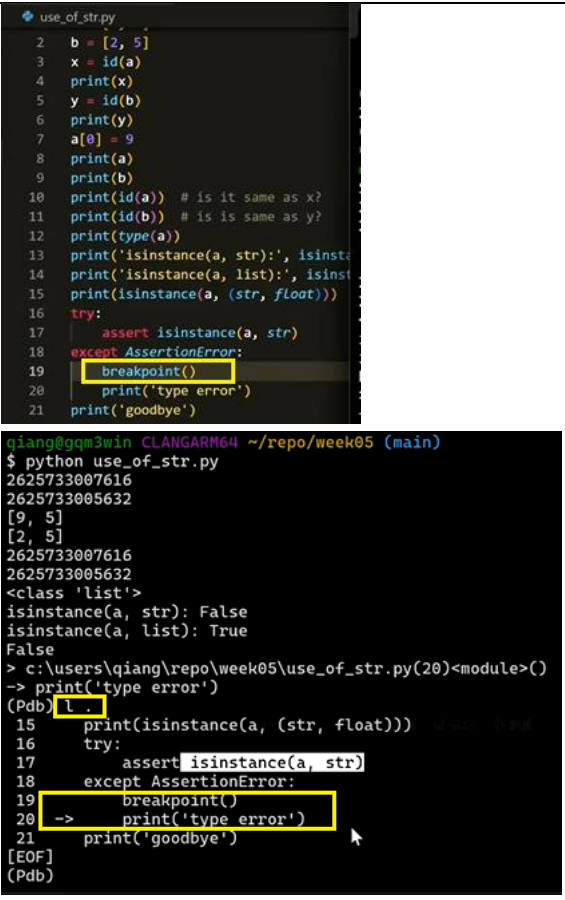
对于任何对象, 都可以传给以下 **内置函数** (built-in function) 用于检视 (inspect):

将 vs code 里的表达式 (expression) 输出到终端, 查看结果是否符合预期	print() 函数	 <pre>! environment.yml U use_o use_of_str.py 1 a = 'hello' 2 print(a) qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py hello</pre>
返回对象在虚拟内存中的地址 (正整数), 如果 id(a) == id(b), 那么 a is b (is 是个运算符)	id()	 <pre>use_of_str.py 1 a = 'hello' 2 x = id(a) 3 print(x) qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py 1519358073216</pre>
返回对象的类型	type()	 <pre>use_of_str.py 1 a = [2, 5] 2 b = [2, 5] 3 x = id(a) 4 print(x) 5 y = id(b) 6 print(y) 7 a[0] = 9 8 print(a) 9 print(b) 10 print(id(a)) # is it same as 11 print(id(b)) # is it same as 12 print(type(a)) qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py 2090626849024 2090626847040 [9, 5] [2, 5] 2090626849024 2090626847040 <class 'list'></pre>
判断对象是否属于某个 (或某些) 类型	isinstance()	 <pre>use_of_str.py 1 a = [2, 5] 2 b = [2, 5] 3 x = id(a) 4 print(x) 5 y = id(b) 6 print(y) 7 a[0] = 9 8 print(a) 9 print(b) 10 print(id(a)) # is it same as x? 11 print(id(b)) # is it same as y? 12 print(type(a)) 13 print(isinstance(a, str)) qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py 2090626849024 2090626847040 [9, 5] [2, 5] 2090626849024 2090626847040 False</pre>

黄色的 isinstance 这些随便写, 只是为了终端输

		<p>出时能对应上，到底谁是 false</p> <p>注意：print 里对象之间要空格分隔</p> <pre>qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py 2673853929728 2673853927744 [9, 5] [2, 5] 2673853929728 2673853927744 <class 'list'> isinstance(a, str): False</pre>
返回对象所支持的属性 (attributes) 的名称列表	dir()	<pre>use_of_str.py 1 a = [2, 5] 2 b = [2, 5] 3 x = id(a) 4 print(x) 5 y = id(b) 6 print(y) 7 a[0] = 9 8 print(a) 9 print(b) 10 print(id(a)) # is it same as x? 11 print(id(b)) # is is same as y? 12 print(type(a)) 13 print('isinstance(a, str): ', isins 14 print('dir(a):', dir(a))</pre> <pre>qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py 2478301255936 2478301253952 [9, 5] [2, 5] 2478301255936 2478301253952 <class 'list'> isinstance(a, str): False dir(a): ['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getstate__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']</pre> <p>同理：</p> <pre>use_of_str.py 1 a = [2, 5] 2 b = [2, 5] 3 x = id(a) 4 print(x) 5 y = id(b) 6 print(y) 7 a[0] = 9 8 print(a) 9 print(b) 10 print(id(a)) # is it same as x? 11 print(id(b)) # is is same as y? 12 print(type(a)) 13 print('isinstance(a, str): ', isinstance(a, str)) 14 print('isinstance(a, list): ', isinstance(a, list)) 15 print('dir(a):', dir(a))</pre> <pre>qiang@gqm3win CLANGARM64 ~/i \$ python use_of_str.py 2095622723840 2095622721856 [9, 5] [2, 5] 2095622723840 2095622721856 <class 'list'> isinstance(a, str): False isinstance(a, list): True</pre> <p>同理：</p> <pre>print('isinstance(a, str): ', isinstance(a, str)) print('isinstance(a, list): ', isinstance(a, list)) print(isinstance(a, str), isinstance(a, list))</pre> <pre>isinstance(a, str): False isinstance(a, list): True True</pre>
返回对象 print 时要显示在终端的字符串	str()	<pre>qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python Python 3.12.9 packaged by conda-forge (main, Mar 4 2025, 22:37:18) [MSC v.1 943 64 bit (AMD64)] on win32 Type "help", "copyright", "credits" or "license" for more information. >>> print(32) 32 >>> print(str(32)) 32 >>></pre>

		Print 就是显示字符串
查验某个表达式(expression) 为真, 否则 报错 (AssertionError) 退出	assert 语句 (查验工具)	<div><pre>use_of_str.py 1 a = [2, 5] 2 b = [2, 5] 3 x = id(a) 4 print(x) 5 y = id(b) 6 print(y) 7 a[0] = 9 8 print(a) 9 print(b) 10 print(id(a)) # is it same as x? 11 print(id(b)) # is it same as y? 12 print(type(a)) 13 print('isinstance(a, str):', isinstance(a, str)) 14 print('isinstance(a, list):', isinstance(a, list)) 15 print('isinstance(a, (str, float)):', isinstance(a, (str, float))) 16 assert isinstance(a, str) 17 print('goodbye')</pre></div> <div><pre>qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py 3001545922816 3001545922832 [9, 5] [2, 5] 3001545922816 3001545922832 3001545922832 <class 'list'> isinstance(a, str): False isinstance(a, list): True False Traceback (most recent call last): File "C:\Users\qiang\repo\week05\use_of_str.py", line 16, in <module> assert isinstance(a, str) AssertionError</pre></div>
把报错 AssertionError 转为 except 后面的内容—— 拦截报错, 避免退出, 流程转移	try 语句 (与 assert 语句联合使用)	<div><pre>use_of_str.py 1 a = [2, 5] 2 b = [2, 5] 3 x = id(a) 4 print(x) 5 y = id(b) 6 print(y) 7 a[0] = 9 8 print(a) 9 print(b) 10 print(id(a)) # is it same as x? 11 print(id(b)) # is it same as y? 12 print(type(a)) 13 print('isinstance(a, str):', isinstance(a, str)) 14 print('isinstance(a, list):', isinstance(a, list)) 15 print('isinstance(a, (str, float)):', isinstance(a, (str, float))) 16 try: 17 assert isinstance(a, str) 18 except AssertionError: 19 print('type error') 20 print('goodbye')</pre></div> <div>AssertionError 要根据实际更改:是根据 assert 后, 报错的内容来更改, 报错提示 AssertionError, 就写 AssertionError 报错提示 indexerror,就写 indexerror</div> <div><pre>qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py 1855538731264 1855538729280 [9, 5] [2, 5] 1855538731264 1855538729280 <class 'list'> isinstance(a, str): False isinstance(a, list): True False type error goodbye (week05)</pre></div>

自动进入 pdb 调试 (debug) 模式，并在 breakpoint() 函数后，暂停代码运行	breakpoint() 函数	 <pre>use_of_str.py 2 b = [2, 5] 3 x = id(a) 4 print(x) 5 y = id(b) 6 print(y) 7 a[0] = 9 8 print(a) 9 print(b) 10 print(id(a)) # is it same as x? 11 print(id(b)) # is it same as y? 12 print(type(a)) 13 print('isinstance(a, str):', isinstance(a, str)) 14 print('isinstance(a, list):', isinstance(a, list)) 15 print('isinstance(a, (str, float)):', isinstance(a, (str, float))) 16 try: 17 assert isinstance(a, str) 18 except AssertionError: 19 breakpoint() 20 print('type error') 21 print('goodbye')</pre> <pre>qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py 2625733007616 2625733005632 [9, 5] [2, 5] 2625733007616 2625733005632 <class 'list'> isinstance(a, str): False isinstance(a, list): True False > c:\users\qiang\repo\week05\use_of_str.py(20)<module>() -> print('type error') (Pdb) l 15 print('isinstance(a, (str, float)):', isinstance(a, (str, float))) 16 try: 17 assert isinstance(a, str) 18 except AssertionError: 19 breakpoint() 20 print('type error') 21 print('goodbye') [EOF] (Pdb)</pre>
---	-----------------	--

2.对于 每一个上述要求掌握的对象类型 (将来遇到新的对象类型也应该如此)，我们首先应该熟悉如何通过 表达式 (expression) 得到他们的实例 (instance)，一般包括以下途径：

- 字面值 (literal) (包括 f-string 语法)
- 推导式 (comprehension) (仅限 list 列表、dict 字典、set 集合)
- 初始化 (init)
- 运算值 (operator)
- 索引值 (subscription)
- 返回值 (return value of function/method call)

	字面值（字面：写啥就是啥）	
	f-string	<pre>print('f-string') x = 'Tom' s = f'name: {x}'</pre>

		<pre> qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py 字面值 university True f-string name: Tom </pre>
	\n 换行	<pre> s = 'aaa\nbbb' print('New Line', s) </pre> <pre> qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py 字面值 university True f-string name: Tom TAB a b New Line aaa bbb </pre>
	"""三对引号：换行 空格：会原样保留	<pre> s = """xyz abc eee aaa """ </pre> <pre> qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py 字面值 university True f-string name: Tom TAB a b New Line aaa bbb xyz abc eee aaa </pre>
	初始化 str() 等于 字面值 '' <code>str() == ''</code>	<pre> print('初始化') s = str() print(s) s = str([5, 8, 2]) print(s) </pre> <p>初始化 [5, 8, 2] 中间空一行</p> <pre> print('初始化') s = str() print(s) s = str([5, 8, 2]) print(s) assert str([5, 8, 2]) == '[5, 8, 2]' assert str(1.1 + 2.2) != '3.3' assert str() == '' </pre> <p>注意：不等于: !=</p> <pre> qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_str.py 字面值 university True f-string name: Tom TAB a b New Line aaa bbb xyz abc eee aaa 初始化 [5, 8, 2] </pre>
	运算值	<pre> s = '=' s = s * 20 print(s) </pre> <pre> ===== </pre>

	索引值	<pre>s = 'hello' assert s[3] == 'l'</pre> <p>【】索引 Hello 是有编号的：01234 取 S 的编号 3 的字母 l</p> <pre>s = 'hello' assert s[3] == 'l' assert s[-1] == 'o' assert s[:3] == 'hel' assert s[4] == s[-1] print(s[5])</pre> <p>: 3 就是取前三个字母 取编号 4 的字母和取编号-1 的字母都是 o 没有编号 5 的字母</p>
	返回值	<p>返回 s 的 upper 值:</p> <pre>s = 'hello' u = s.upper() print(u) print(s)</pre> <pre>HELLO hello</pre> <p>返回 s 的 format 值:</p> <pre>t = 'name: {}, age {}'.format('Jack', 21) print(t) t1 = t.format('Jack', 21) print(t1)</pre> <pre>name: {}, age {} name: Jack, age 21</pre>

3.对于 每一个 上述要求掌握的对象类型 (将来遇到新的对象类型也应该如此)，我们也要尝试验证其以下几个方面的 属性 (attributes):

对数学运算符 (+、-、*、**、/、//、%、@) 有没有支持

如何判断相等 (==)

对于比较运算符 (>、<、>=、<=) 有没有支持

什么值被当作 True，什么值被当作 False

是否可迭代 (iterable)，如何做迭代 (for 循环)

是否支持返回长度 (len)

是否 (如何) 支持索引操作 (subscription) ([] 运算符)

拥有哪些常用方法 (method) 可供调用 (() 运算符)

建议先在 pdb 里试验，然后把确定能够运行的代码写在 use_of_{name}.py 文件里

	数学运算符 (+、-、*、**、	+与 - (下面的例子：可以加，不可以减)
--	---------------------	-----------------------

	<p>/、//、%、@)</p> <p>有没有支持</p>	<pre>s1 = 'abc' s2 = 'ghi' s = s1 + s2 assert s == 'abcghi' print(s2 + s1) print(s2 - s1)</pre> <pre>ghiabc Traceback (most recent call last): File "C:\Users\qiang\repo\week05\use_of_str.py", line 71, in <module> print(s2 - s1) TypeError: unsupported operand type(s) for -: 'str' and 'str'</pre> <p>*与%（下面的例子：可以乘，不可以除）</p> <pre>s = ' *= ' s = s * 10 print(s) s = 'aaaa' s = s / 2</pre> <pre>Traceback (most recent call last): File "C:\Users\qiang\repo\week05\use_of_str.py", line 81, in <module> s = s / 2 TypeError: unsupported operand type(s) for /: 'str' and 'int'</pre>
	<p>判断相等</p> <p>(==)</p>	<pre>assert s == 'aaaa'</pre> <p>不报错</p>
	<p>比较运算符</p> <p>(>、<、>=、<=)</p>	 <p>所有的字符都在上表里，有编码</p> <p>所以字符可以比较大小：</p> <pre>print('abc' > 'ABC') print('123' < 'abcd') print('9' > '.')</pre> <p>True True True</p> <p>abc 在表里的位置靠后，编码数字更大，所以 abc>ABC</p> <p>字符串也可以比较大小：</p> <p>规则是先比较第一个字符，大，就是大于，后面不用看，如果相等在比较第二个字符，以此类推。</p>
	<p>什么值被当作 True，什么值被当作 False</p>	<p>只要这个字符串的长度。不是零，那它都是 true 长度为零，这个字符串长度为零，它就是 false。</p> <pre>assert 'book' assert ''</pre> <pre>True Traceback (most recent call last): File "C:\Users\qiang\repo\week05\use_of_str.py", line 96, in <module> assert '' AssertionError</pre>
字符串是可以做循环的，然	<p>是否可迭代</p> <p>(iterable)</p> <p>如何做迭代</p>	<p>Iter: 能否迭代</p> <pre>s = 'book' print(iter(s))</pre> <pre>True <str ascii iterator object at 0x000001F5C7930F70></pre>

后循环就是对这个字符串里的每一个字符挨个的跑一遍。	(for 循环)	<p>For</p> <pre> for c in s: print(c) book </pre>
	长度 (len)	<pre> print(len(s)) 4 </pre>
		<pre> s = 'book' assert s[1:3] == "oo" </pre> <p>不报错</p> <p>这个是固定的规则，就是 python 里面这个前面这个是包含的，后边这个是不包含的，所以 1: 3 是指编号为一的字符到编号为 3 的字符，包含 1 不包含 3。</p>
	查看内置变量的使用方法 (method)	<p>先在 main.py 的代码里加一行 <code>108 breakpoint()</code></p> <p>然后终端输入：<code>qiang@gqm3win CLANGARM64 ~/repo/week05 (main)</code> <code>\$ python use_of_str.py</code></p> <p>自动进入 pdb 然后停在 breakpoint 处：</p> <pre> CLANGARM64 x/qiang x + v 初始化 [s, 8, 2] ===== string index out of range HELLO hello name: {}, age {} name: 'jack', age 21 ghiabc unsupported operand type(s) for -: 'str' and 'str' ===== unsupported operand type(s) for /: 'str' and 'int' True True True True True True <str_ascii_iterator object at 0x000011EFB311120> b o o k 4 --Return-- > c:\users\qiang\repo\week05\use_of_str.py(108)<module>()->None -> breakpoint() (Pdb) </pre> <p>终端输入 l: 看运行到哪里了</p> <pre> -> breakpoint() (Pdb) l 103 104 print(len(s)) 105 106 s = 'book' 107 assert s[1:3] == "oo" 108 -> breakpoint() [EOF] (Pdb) </pre> <p>对照：<pre>s = 'book' assert s[1:3] == "oo" breakpoint()</pre> 发现我这里的 s 不太明白，或者我想知道 S 的各种属性的使用方法，那我就：</p> <pre> (Pdb) p s 'book' (Pdb) import wat (Pdb) wat / s </pre> <p>出现：内置变量 s 的各种属性 attributes 的使用方法(如：capitalize())</p>

		<pre>[EOF] (Pdb) p s 'book' (Pdb) import wat (Pdb) wat / s value: 'book' type: str len: 4 Public attributes: def capitalize() # Return a capitalized version of the string... def casefold() # Return a version of the string suitable for caseless comparisons. def center(width, fillchar=' ') # Return a centered string of length width. ... def count(sub[, start[, end]]) -> int... def encode(encoding='utf-8', errors='strict') # Encode the string using the codec registered for encoding... def endswith(suffix[, start[, end]]) -> bool... def expandtabs(tabsize=8) # Return a copy where all tab characters are expanded using spaces... def find(sub[, start[, end]]) -> int... def format(*args, **kwargs) -> str... def format_map(mapping) -> str... def index(sub[, start[, end]]) -> int... def isalnum() # Return True if the string is an alpha-numeric string, False otherwise...</pre>
		<p>比如我想知道 count 属性的使用方法</p> <pre>... def count(sub[, start[, end]]) -> int...</pre> <p>你就：</p> <p>Wat 空格/空格 变量 . 变量的某属性</p> <pre>(Pdb) wat / s.count value: <built-in method count of str object at 0x00000248A32FE8F0> type: builtin_function_or_method signature: def count(sub[, start[, end]]) -> int """ S.count(sub[, start[, end]]) -> int Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation. """</pre>

(二) 字节串 bytes

字节串 00111000 解码后，变回字符 abc

字符串 abc 编码后，变成字节 00111000

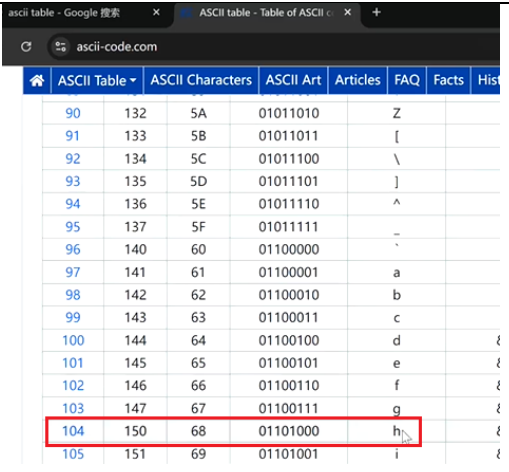
```
p = Path("C:\\Users\\qiang\\anaconda3\\envs\\week05\\python.exe")
s = p.read_bytes()
print(len(s))

p = Path("environment.yml")
b = p.read_bytes()
print(b[0])

s = b.decode()
assert isinstance(s, str)
b2 = s.encode()
assert isinstance(b2, bytes)
assert b2 == b
```

无报错

		<pre>use_of_bytes.py > ... 1 s = b"hello" 2 print(s) 3 print(s[0]) 4</pre> <pre>b'hello' 104</pre> <p>之所以出来 104，是因为： 字符 h 对应的编号是 104，他的字节是 01100111</p>
--	--	--

		
--	--	--

(三) 整数 int

	//整除 %余除	<pre>6 x = 5 7 y = 17 8 assert y // x == 3 9 assert y % x == 2</pre> <p>17=5*3+2</p>
--	-------------	--

(四) 浮点数 float

	字面值 3.14 Type: float	<pre>x = 3.14 print(type(x))</pre> <pre>qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_float.py <class 'float'></pre>
	初始化字符串为浮点数	<p>小数可以初始化为浮点数</p> <pre>y = float("3.14") print(type(y))</pre> <pre>qiang@gqm3win CLANGARM64 ~/repo/week05 (main) \$ python use_of_float.py <class 'float'></pre> <p>字母字符不可以:</p> <pre>f = float("hello")</pre> <pre>Traceback (most recent call last): File "C:\Users\qiang\repo\week05\use_of_float.py", line 19, in <module> f = float("hello") ValueError: could not convert string to float: 'hello'</pre>
	计算	<p>整数除乘数，除不尽=浮点数</p> <pre>x = 5 / 3 print(x, type(x))</pre> <pre>1.6666666666666667 <class 'float'></pre> <p>0-1 随机去数，也是浮点数:</p> <pre>x = random.random() print(x)</pre> <pre>0.16996318389429454</pre> <p>特殊地 1: nan 表示为缺失值，缺失值（初始化为浮点数后）与任何数计算都是缺失值</p>

		<pre>nan = float("nan") print(nan + 3) print(nan > 3) print(nan < 3) print(nan == 3)</pre> <p>特殊地 2: inf 表示为正无穷大, 正无穷 (初始化为浮点数后) 大于任何一个浮点数</p> <pre>1e200</pre> <p>=1*10 的 200 次方</p> <pre>pinf = float("inf") print(3.14e-2) print(pinf > 1e200)</pre> <pre>0.0314 True</pre> <p>特殊地 3: 同理-inf 为负无穷</p> <pre>ninf = float("-inf") print(ninf)</pre>
	什么值被当作 True, 什么值被当作 False	<p>0.0 是 false</p> <pre>assert 0.0</pre> <pre>Traceback (most recent call last): File "C:\Users\qiang\repo\week05\use_of_float.py", line 17, in <module> assert 0.0 *** AssertionError</pre>
	判断“等不等”	<p>因为浮点数是自动四舍五入的数, 会有误差, 计算机无法判断等不等</p>

(五) 布尔值 (bool)

布尔, 它是一种特殊的整数, 就仅限于零和一这样的一种整数。true 就当做是 1, false 就当做是 0。

	字面值	<pre>t = True f = False print(t, f)</pre> <pre>True False</pre>
	是一种整数	<pre>print(isinstance(t, int))</pre> <pre>True</pre>

(六) 列表 (list)

四个容器——列表 (list)、字典 (dict)、元组 (tuple)、集合 (set)

	<p>字面值</p> <p>与索引【】</p> <p>与编号 0、1、2、3</p>	<pre>l = [1, 5, "abc"] print(l) print(l[0]) print(l[1]) print(l[2]) print(l[3])</pre> <pre>[1, 5, 'abc'] 1 5 abc Traceback (most recent call last): File "C:\Users\qiang\repo\week05\use_of_list.py", line 8, in <module> print(l[3]) *** IndexError: list index out of range</pre> <pre>l = [1, 5, "abc"] print(l) print(l[0]) print(l[1]) print(l[2]) try: print(l[3]) except IndexError as e: print(e) print(l[-1]) print(l[-1][1])</pre>
--	--	---

		索引里的索引 <pre>1, 5, 'abc'] 1 5 abc list index out of range abc b</pre>
	计算（加减乘除）	<pre>a = [2, 5] b = ["a", "c"] print(a + b) print(b + a) print(a + b == b + a) a = [2, 5] b = [5] print(a - b)</pre> <pre>[2, 5, 'a', 'c'] ['a', 'c', 2, 5] False Traceback (most recent call last): File "C:\Users\qiang\repo\week05\use_of_list.py", line 24, in <module> print(a - b) TypeError: unsupported operand type(s) for -: 'list' and 'list'</pre> 乘法: <pre>a = [2, 5] print(a * 3)</pre> <pre>[2, 5, 2, 5, 2, 5]</pre>
		<pre>a = [2, 5] b = a * 3 print(f"{b=}") a[0] = 9 print(a) print(b)</pre> <pre>b=[2, 5, 2, 5, 2, 5] [9, 5] [2, 5, 2, 5, 2, 5]</pre> <pre>a = [2, 5] b = [a] * 3 print(f"{b=}") a[0] = 9 print(a) print(b)</pre> <pre>b=[[2, 5], [2, 5], [2, 5]] [9, 5] [[9, 5], [9, 5], [9, 5]]</pre> 例子 3: append 合并函数: 修改这个列表, 往这个列表后, 加一个东西 <pre>a = [2, 5] b = [a] * 3 print(f"{b=}") x = a.append(4) print(x) print(a) print(b)</pre> <pre>b=[[2, 5], [2, 5], [2, 5]] None [2, 5, 4] [[2, 5, 4], [2, 5, 4], [2, 5, 4]]</pre>
	推导式（列表的衍生品）——很常用	例子 1: <pre>a = [2, 5, 3] b = [i*2 for i in a] print(b)</pre> <pre>[4, 10, 6]</pre> 例子 2: 带 if 条件 <pre>a = [2, 5, 3] b = [i*2 for i in a] print(b) b = [i*2 for i in a if i < 4] print(b)</pre> <pre>[4, 10]</pre> 5 不符合要求, 直接跳过, 只写符合要求的

（七）字典（dict）——python 里最最最重要的类型

字典与列表的区别:

列表和字典都是容器, 但列表有编号, 字典没编号, 字典需要你给他命名, 通过调用命名来找到对应的值, 列表是通过编号（按照整数 0123）找到对应的值。

键值对: a 为“键”, 1 为值, () 元组, 一个键值对构成一个元组

```
[('a', 1), ('bb', 5), ('cat', 3)]
```

	字典面值	<pre>d = {"a": 1, "bb": 5, "cat": 3}</pre> 上面 {} 就是一个字典
--	------	---

		<p>没有编号</p> <p>调用的时候，直接告诉他你 a、bb 等</p> <p>原理是：a 对应一个哈希值，你问 a 他就会给你找哈希值对应的东西，然后返回给你</p> <pre>(Pdb) p hash('a') -1322551479446760802</pre>
	循环	<p>做 d 的循环</p> <p>循环出“键”</p> <pre>for a in d: print(a)</pre> <pre>a bb cat</pre> <p>循环出“键”的值（两个方法）</p> <pre>for a in d: print(d[a]) for a in d.values(): print(a)</pre> <pre>1 5 3 1 5 3</pre> <p>循环出“键”+值的键值对（“元组”），Item 出来的都是元组（两个方法）</p> <pre>l = [a for a in d.items()] print(l)</pre> <pre>[('a', 1), ('bb', 5), ('cat', 3)]</pre> <pre>for k, v in d.items(): print(k, v)</pre> <pre>a 1 bb 5 cat 3</pre>
	什么值被当作 True，什么值被当作 False	<p>False: 空的字典</p> <p>True: 只要不是空的</p> <pre>assert {}</pre> <pre>Traceback (most recent call last): File "C:\Users\qiang\repo\week05\use_of_dict.py", line 20, in <module> assert {} ^^ AssertionError</pre>

（八）元组 (tuple)

元组和列表的共性：

都可以通过编号（按照整数 0123）找到对应的值。

元组和列表的区别：

元组，是不可以修改的，是不可变的对象：不支持赋值

	不支持赋值	<pre>t[0] = 9</pre> <pre>Traceback (most recent call last): File "C:\Users\qiang\repo\week05\use_of_tuple.py", line 9, in <module> t[0] = 9 ~~~~ TypeError: 'tuple' object does not support item assignment</pre> <p>不报错，并把错误信息显示出来</p>
--	-------	---

		<pre> t = (1, "a", 3.14) print(t) print(type(t)) print(t[0]) print(t[1]) print(t[2]) try: t[0] = 9 except TypeError as e: print(e) </pre>
	<p>字典：</p> <p>字典里的键必须是不可变的对象，下面以给字典的键赋值为例：</p> <p>因为列表是可变的</p> <p>！所以列表不可以做为字典里的键</p> <p>因为元组是不可变的</p> <p>！所以元组可以作为字典里的键</p> <p>可以通过“列表”，变相赋值</p>	<p>1.给字典设置键值对：</p> <pre> d = {} d["abc"] = 5 print(d) </pre> <p>D 是字典</p> <p>给 d 设置一个键，值为 5，那么打印出来就是如下：</p> <pre> {'abc': 5} </pre> <p>2. ！ 列表不可以做为字典里的键：</p> <pre> d = {} d["abc"] = 5 d[7] = 100 q = [3, 1] d[q] = 21 print(d) </pre> <pre> 'tuple' object does not support item assignment Traceback (most recent call last): File "C:\Users\qiang\repo\week05\use_of_tuple.py", line 18, in <module> d[q] = 21 ~~~~^ TypeError: unhashable type: 'list' </pre> <p>3. ！ 元组可以作为字典里的键：</p> <pre> d = {} t = (3, 1) d[t] = 21 print(d) </pre> <pre> {'abc': 5, 7: 100, (3, 1): 21} </pre>
	元组符号可以省略，如果输出值没有歧义的话	<pre> t = 1, 4, 0, 2 print(t) print(type(t)) </pre>

（九）集合（set）

集合和字典的共性：

符号都是{}；无序

集合和字典的区别：

集合只有键，没有值

字典键值都有

	<p>Set 和 {} 的作用相同=集合</p> <p>集合性质 1：唯一性，不重复</p>	<pre> q = [1, 2, 1, 2, 5, 1] print(q) s = set(q) print(s) </pre> <pre> {1, 2, 5} </pre> <pre> s = {5, 2, 1, 2, 1} print(s) </pre>
--	--	--

		<code>{1, 2, 5}</code>
	判断在不在集合里： 用 in	<pre>s = {5, 2, 1, 2, 2, 1} print(s) print(2 in s) print(3 in s)</pre> <div>True False</div>
	方法	<pre>(Pdb) wat / s value: {1, 2, 5} type: set len: 3 Public attributes: def add(...) # Add an element to a set... def clear(...) # Remove all elements from this set. def copy(...) # Return a shallow copy of a set. def difference(...) # Return the difference of two or more sets def difference_update(...) # Remove all elements of another set def discard(...) # Remove an element from a set if it is a memb def intersection(...) # Return the intersection of two sets as def intersection_update(...) # Update a set with the intersecti another. def isdisjoint(...) # Return True if two sets have a null inter def issubset(other, /) # Test whether every element in the se def issuperset(other, /) # Test whether every element in othe def pop(...) # Remove and return an arbitrary set element... def remove(...) # Remove an element from a set; it must be a me def symmetric_difference(...) # Return the symmetric difference new set... def symmetric_difference_update(...) # Update a set with the sy e of itself and another. def union(...) # Return the union of sets as a new set... def update(...) # Update a set with the union of itself and oth</pre>
	集合内部无序，和字典一样	
	计算：并集 、交集&、对称差^ 没有全集，所以也没补集	<pre>s = {5, 2, 1, 2, 2, 1} print(s) print(2 in s) print(3 in s)</pre> <pre>s2 = {3, 2, 3} print(s s2) print(s & s2) print(s ^ s2)</pre> <div>{1, 2, 3, 5} {2} {1, 3, 5}</div> <p>并集 交集& 对称差^</p> 

(十) pathlib

Python 标准库 (standard library) 里的 `pathlib` 和 `datetime` 模块 (module) 提供了用于处理 **路径** 和 **日期时间** 的类型，也是非常基础、非常常用的。标准库模块都不需要安装 (pip/conda install)，但使用前需要导入 (import)。

	使用前需要导入 (import)	<code>from pathlib import Path</code>
	字面值	<pre>p = Path(".") print(p)</pre> <div>.</div>

		注意: .代表当前目录
	方法	<p>以变量 <code>p</code>, 方法 <code>absolute()</code>为例,</p> <p>介绍如何用 <code>pdb</code> 查找一个变量的使用方法, 以及如何在 <code>vscode</code> 里使用方法</p> <p>在 <code>vscode</code> 里正在写代码, 然后在某一行加上 <code>breakthrough</code></p> <p>然后在终端输入: <code>Python +代码所在文件名</code></p> <p>然后终端输入 <code>import wat</code> 和 <code>wat / 变量 p</code>, 出现绿色的方法, 查看灰色的方法说明,</p> <pre>(Pdb) wat / p str: . repr: WindowsPath('.') type: pathlib.WindowsPath parents: pathlib.Path, pathlib.PureWindowsPath, pathlib.PurePath Public attributes: anchor: str = '' drive: str = '' name: str = '' parent: pathlib.WindowsPath = . parents: pathlib.PathParents = <WindowsPath.parents> parts: tuple = () root: str = '' stem: str = '' suffix: str = '' suffixes: list = [] def absolute() # Return an absolute version of this path by def as_posix() # Return the string representation of the pa def as_uri() # Return the path as a 'file' URI. def chmod(mode, *, follow_symlinks=True) # Change the perm def cwd() # Return a new path pointing to the current worki</pre> <p>如果说还没看懂 <code>absolute()</code>方法, 那就:</p> <p>终端输入 <code>p.p绝对</code></p> <p>然后在 <code>vscode</code> 使用这个方法, 如使用变量 <code>p</code> 的 <code>absolute()</code>方法: <code>print(p.absolute())</code></p> <pre>from pathlib import Path p = Path(".") print(p) print(p.exists()) print(p.absolute())</pre> <p>得到返回值:</p> <pre>C:\Users\qiang\repo\week05</pre>
	比较有用的方法	<p>1:</p> <pre>def iterdir() # Yield path objects of the directory contents...</pre> <p>列举文件夹里边所有的东西</p> <pre>from pathlib import Path from pprint import pprint p = Path(".") print(p) print(p.exists()) print(p.absolute()) pprint(list(p.iterdir()))</pre>

		<pre>[WindowsPath('.git'), WindowsPath('.gitignore'), WindowsPath('environment.yml'), WindowsPath('LICENSE'), WindowsPath('README.md'), WindowsPath('use_of_bool.py'), WindowsPath('use_of_bytes.py'), WindowsPath('use_of_dict.py'), WindowsPath('use_of_float.py'), WindowsPath('use_of_int.py'), WindowsPath('use_of_list.py'), WindowsPath('use_of_path.py'), WindowsPath('use_of_set.py'), WindowsPath('use_of_str.py'), WindowsPath('use_of_tuple.py')] 2: def mkdir(mode=511, parents=False, exist_ok=False) # Create a new directory</pre>	

(十一) datetime

Python 标准库 (standard library) 里的 pathlib 和 datetime 模块 (module) 提供了用于处理 **路径** 和 **日期时间** 的类型，也是非常基础、非常常用的。标准库模块都不需要安装 (pip/conda install)，但使用前需要导入 (import)。

	使用前需要导入 (import)	<pre>from datetime import datetime, date, timedelta</pre> <p>这三个都是类型</p>	
	显示日期	<pre>print(date.today())</pre> <pre>2025-04-07</pre>	
	计算：加减法	<pre>t1 = date.today() t2 = date(2025, 11, 11) td = t2 - t1 print(td) print(type(td))</pre> <pre>218 days, 0:00:00 <class 'datetime.timedelta'></pre> <p>拓展：date 的属性 days:</p> <pre>t1 = date.today() t2 = date(2025, 11, 11) td = t2 - t1 print(td) print(type(td)) print(td.days)</pre> <pre>218</pre>	
	datetime	<pre>def strftime(...) # format -> strftime() style string</pre>	

		<pre>s1 = "2024-05-23" s2 = "2024-12-04" d1 = datetime.strptime(s1, "%Y-%m-%d") d2 = datetime.strptime(s2, "%Y-%m-%d") print(d1) print(d2) breakpoint()</pre> <pre>2024-05-23 00:00:00 2024-12-04 00:00:00</pre>	
--	--	--	--

