day01 课堂笔记

课程之前

课程介绍

```
目标:
1. 学习掌握 python 的基本语法
2. 在代码中遇到常见的错误,自己能够动手解决问题

Python 只是一个编程语言,在工作中需要结合其他的工具使用
Python + selenium  web 自动化(功能测试转换为代码)
Python + appium    移动端(手机端 APP)自动化
Python + requests   接口
```

1	Python基础	1. 认识Python 2. Python环境搭建 3. PyCharm 4. 注释、变量、变量类型、输入输出、运算符
2	流程控制结构	1. 判断语句 2. 循环
3	数据序列	1. 字符串 2. 列表 3. 元组 4. 字典
4	函数	1. 函数基础 2. 变量进阶 3. 函数进阶 4. 匿名函数
5	面向对象	1. 面向对象编程介绍 2. 类和对象 3. 面向对象基础语法 4. 封装、继承、多态 5. 类属性和类方法
6	异常、模块、文件操作	1. 异常 2. 模块和包 3. 文件操作
7	UnitTest框架	1. UnitTest基本使用 2. UnitTest断言 3. 参数化 4. 生成HTML测试报告

今日内容

1. 了解Python语言及其应用领域
2. 了解Python运行原理
3. 掌握如何安装Python解释器
4. 知道如何安装PyCharm
5. 掌握如何在PyCharm中编写Python代码并运行
6. 掌握单行注释和多行注释的使用方式
7. 掌握变量的使用
8. 掌握常见的数据类型
9. 熟悉常用的运算符

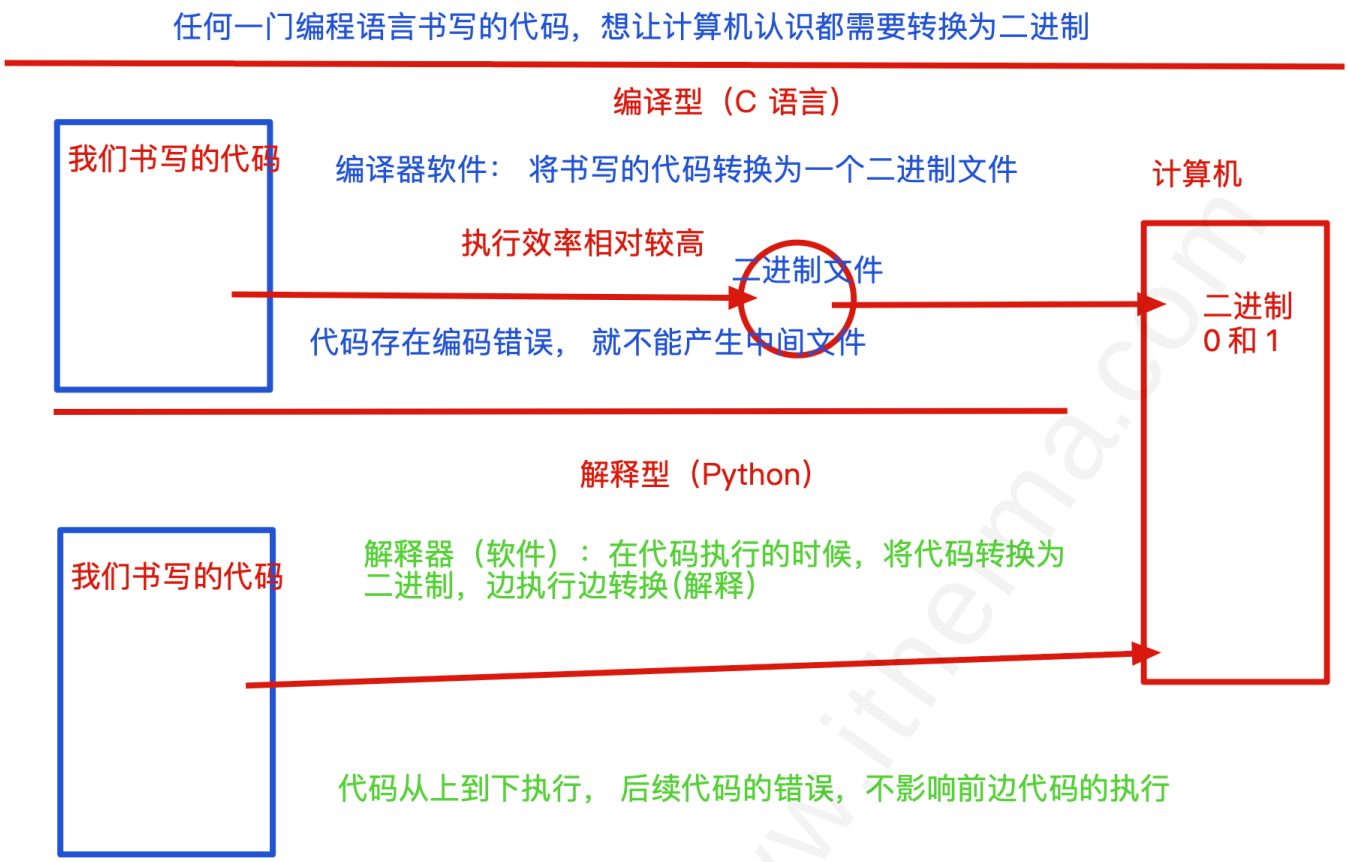
Python 介绍[了解]

```
作者：吉多·范罗苏姆(Guido van Rossum) 龟叔
1989 年开始书写，1991年诞生
```

- 问什么学习 Python?
  - 简单, 易学, 免费, 开源, 适用人群广泛
  - 应用领域广泛(自动化测试)
- Python 的版本
  - Python2 (2.x 2.7)
  - Python3(主流使用的版本, 3.6 之后的版本(即大于等于 3.6))

语言的分类

```
计算机只认识 二进制(0 和 1).
编程语言是人和计算机沟通的语言.
编程语言分类：编译型语言，解释型语言
```



Python 环境配置

python 解释器(必须有): 将我们书写的 Python 代码转换为二进制, 建议 版本 >= 3.6

pycharm(选装): 是 Python 中最好用的IDE(集成开发环境)之一, 是用来书写代码运行代码,调试代码的...

vscode, idle , 记事本 ...

Python 解释器的安装

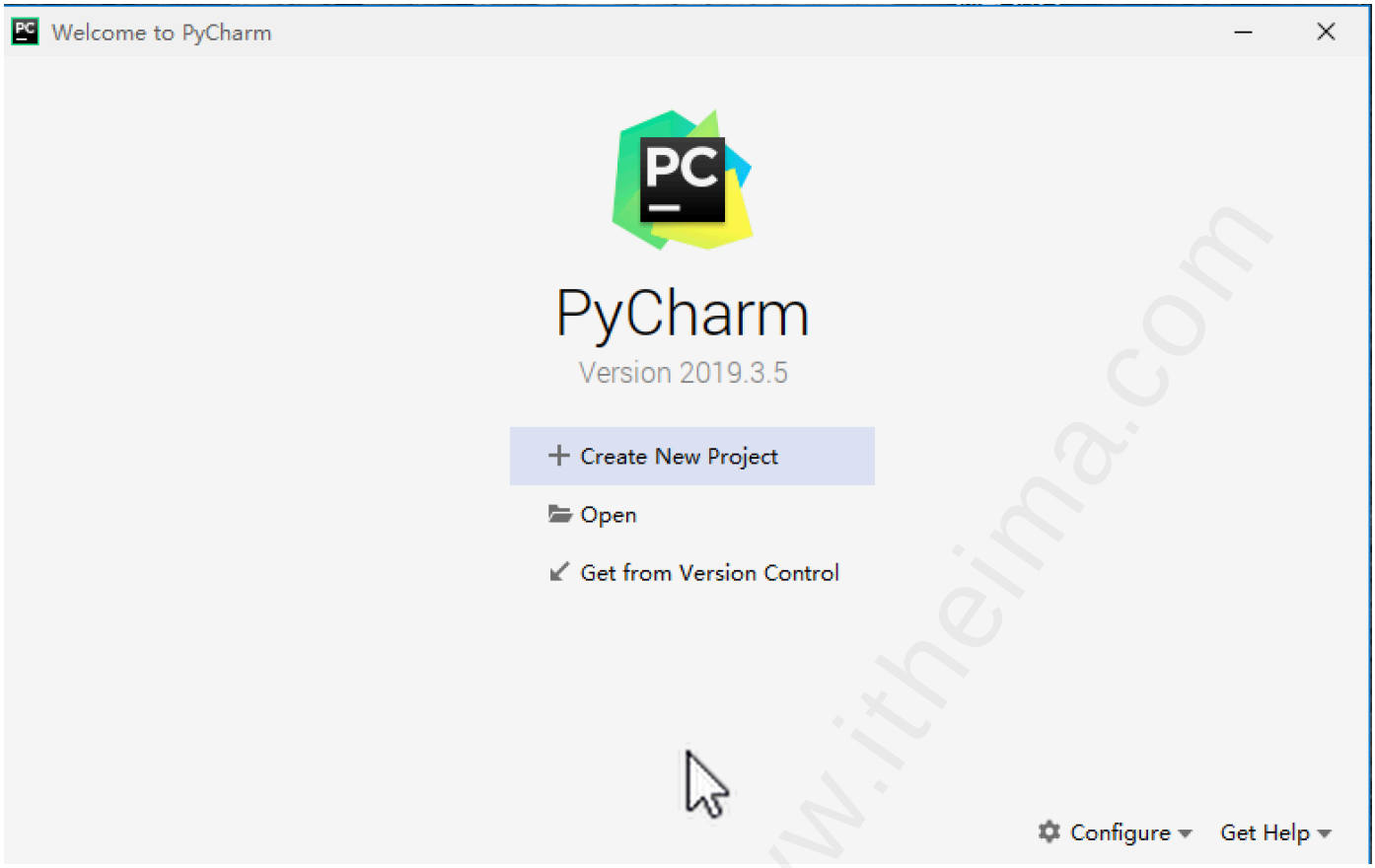
1. 双击安装包
2. 选择 安装方式(可以默认安装, 可以自定义), 不要忘了 勾选 `添加path环境变量`

> path 环境变量：将一个软件或者程序添加到 path 环境变量之后，就可以使用终端(cmd) 在任意路径下使用 这个软件(如果没有环境变量,只能在安装目录使用)
>
> Python 安装之后，在桌面中没有快捷方式,也不需要，是在 pycharm 中使用或者在cmd 中使用

pycharm 的配置安装

> pycharm 有两个版本，一个是专业版(收费的)，一个社区版(免费使用)

直接双击安装即可, 看见一下界面即可



路径的选择(建议)

1. 可以直接使用默认的路径
2. 自定义路径

2.1 不建议使用中文

2.2 可以在某个盘的根目录中创建一个目录 tools，可以将所以学习阶段的环境都安装在 tools 目录

2.3 Python 安装，tools 目录中创建 Python36 目录，pycharm 安装,创建 pycharm 的目录，其他软件的安装,都创建一个目录

3. 严禁安装之后,自己剪切移动目录

使用 pycharm 书写代码

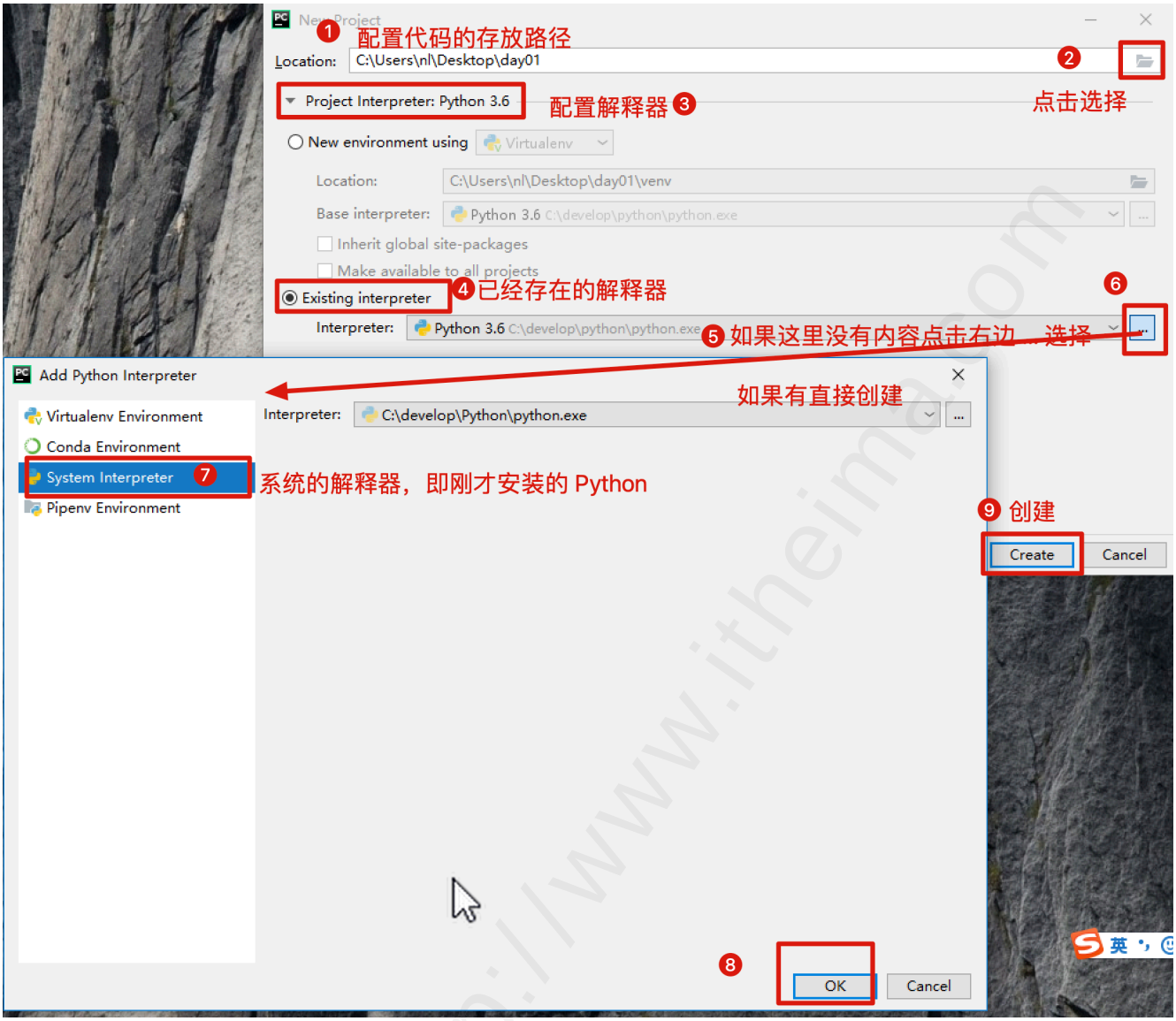
pycharm 是书写代码的软件,还能运行代码，运行代码的前提是在 pycharm 软件中配置了解释器.

pycharm 组织代码的方式是 项目(project)，简单的理解为一个目录，目录中可以放很多的代码

建议：每天的代码作为一个项目
每次在创建项目的时候，需要保证这个目录是一个空目录

1. 双击打开 `pycharm` 软件
2. 创建新 项目

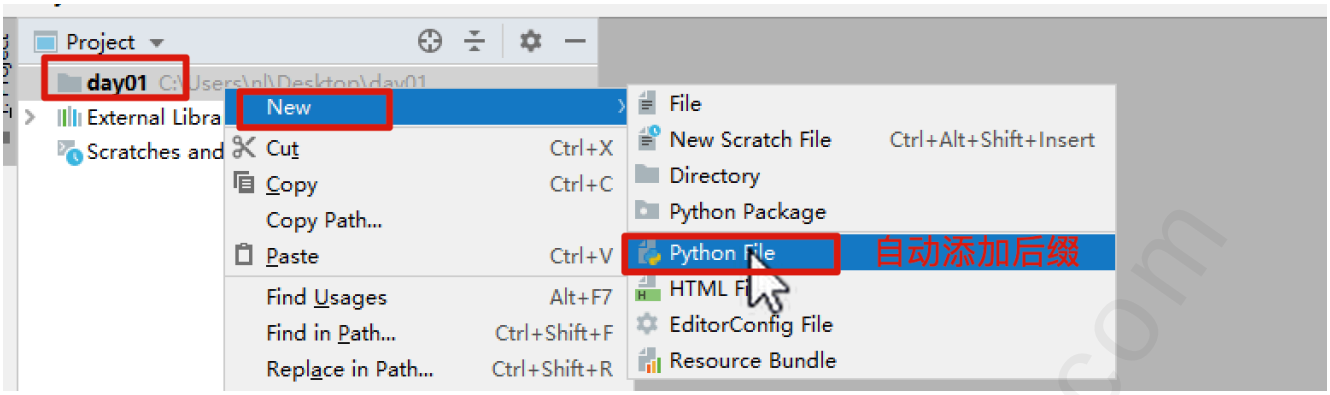
3. 配置项目的路径和解释器

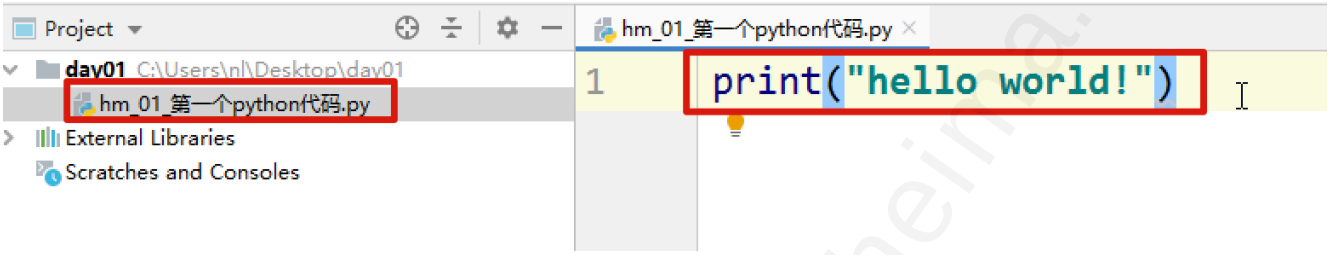


4. 创建代码文件书写代码

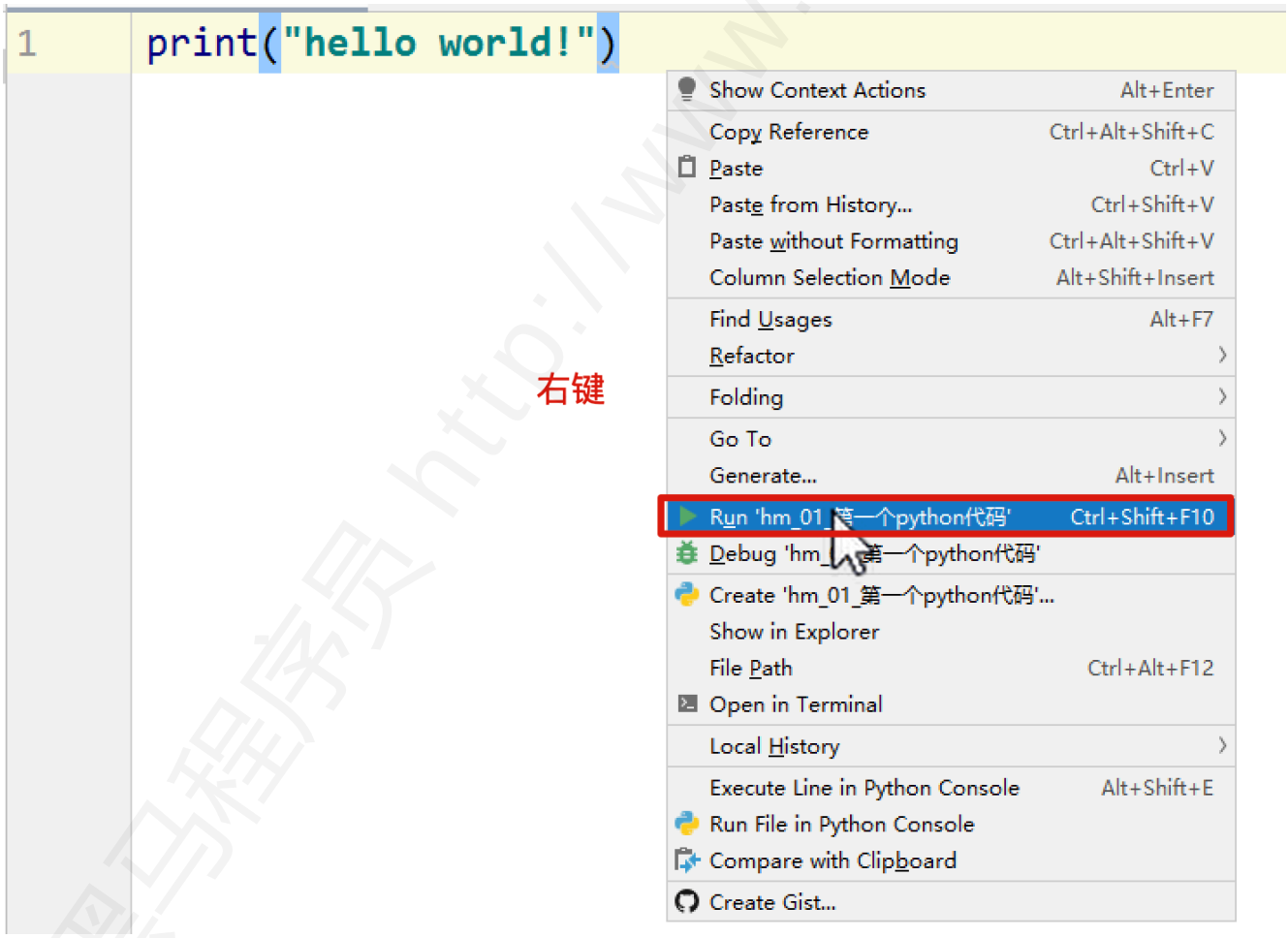
1. 将来在工作中，代码的文件名字不要使用中文,但目前学习阶段,我会使用中文
2. Python 文件的后缀是 .py

3. 代码要顶格书写
4. 代码中的标点符号要使用英文状态的标点

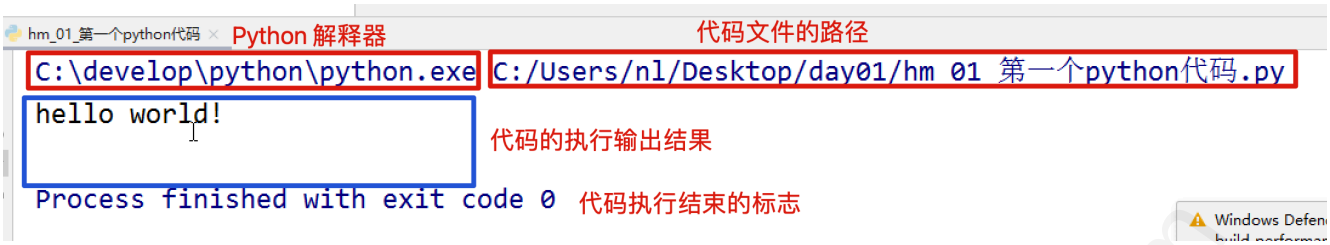




5. 运行代码文件

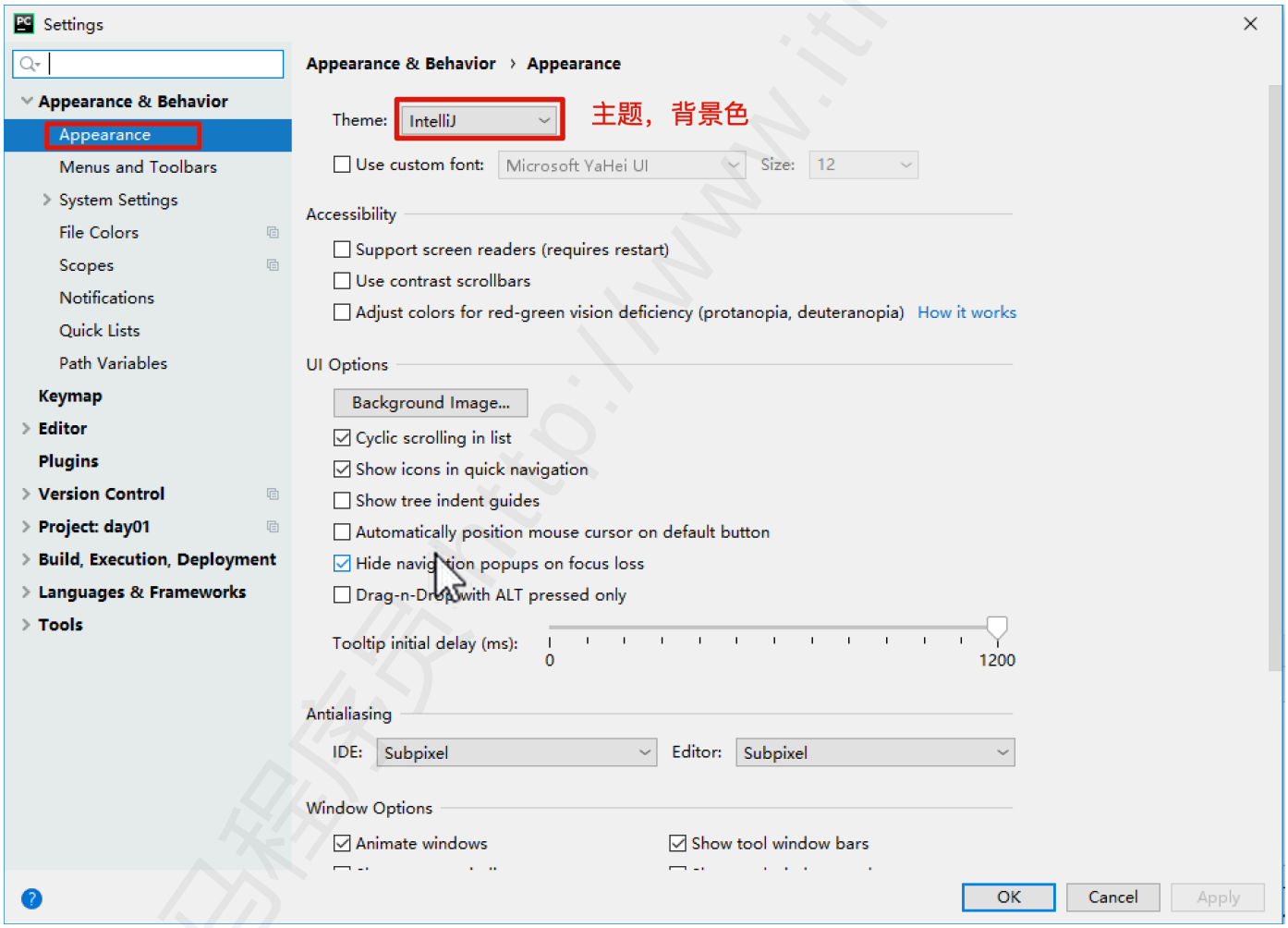


6. 查看运行结果

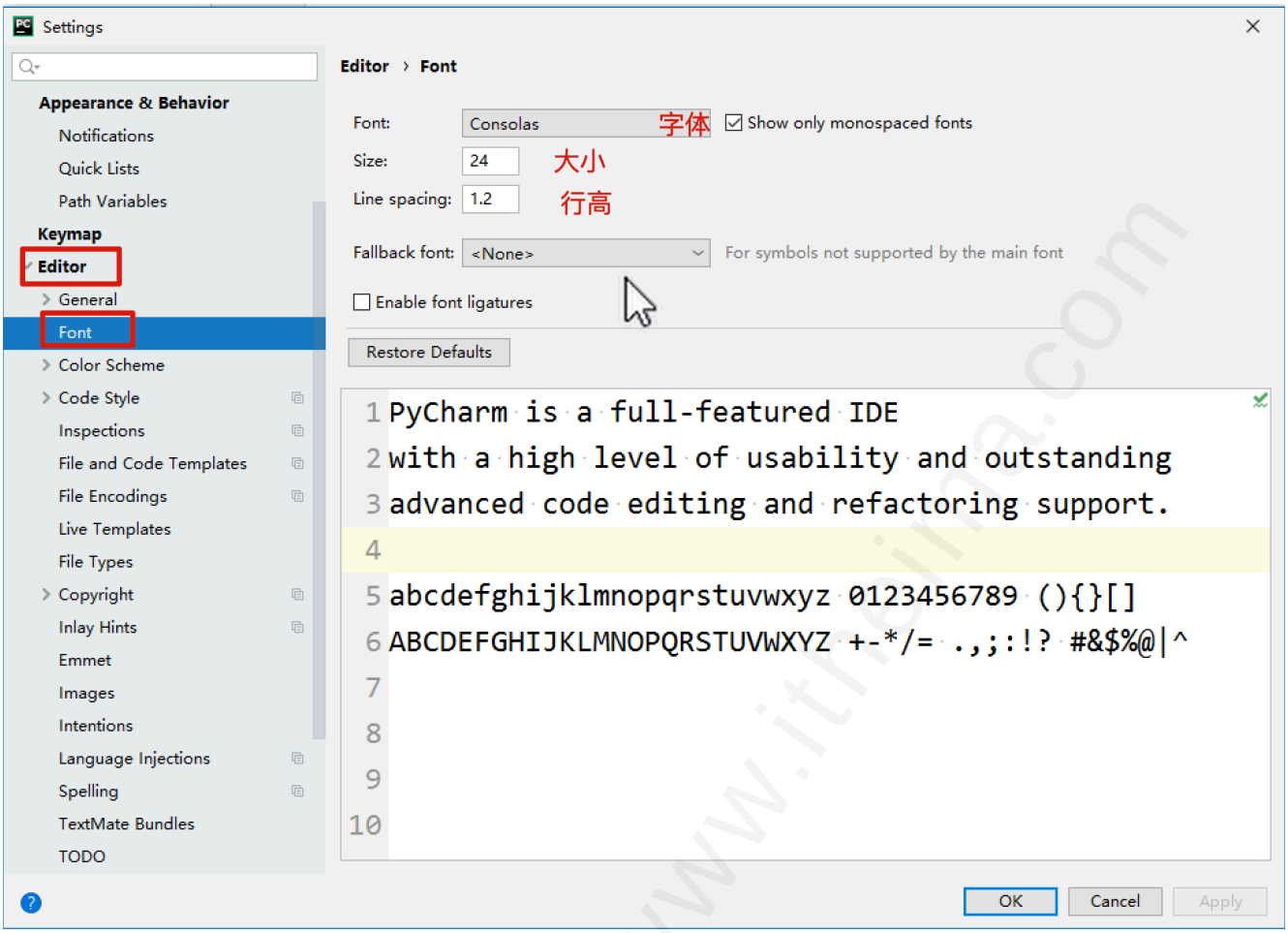


pycharm 常见的设置

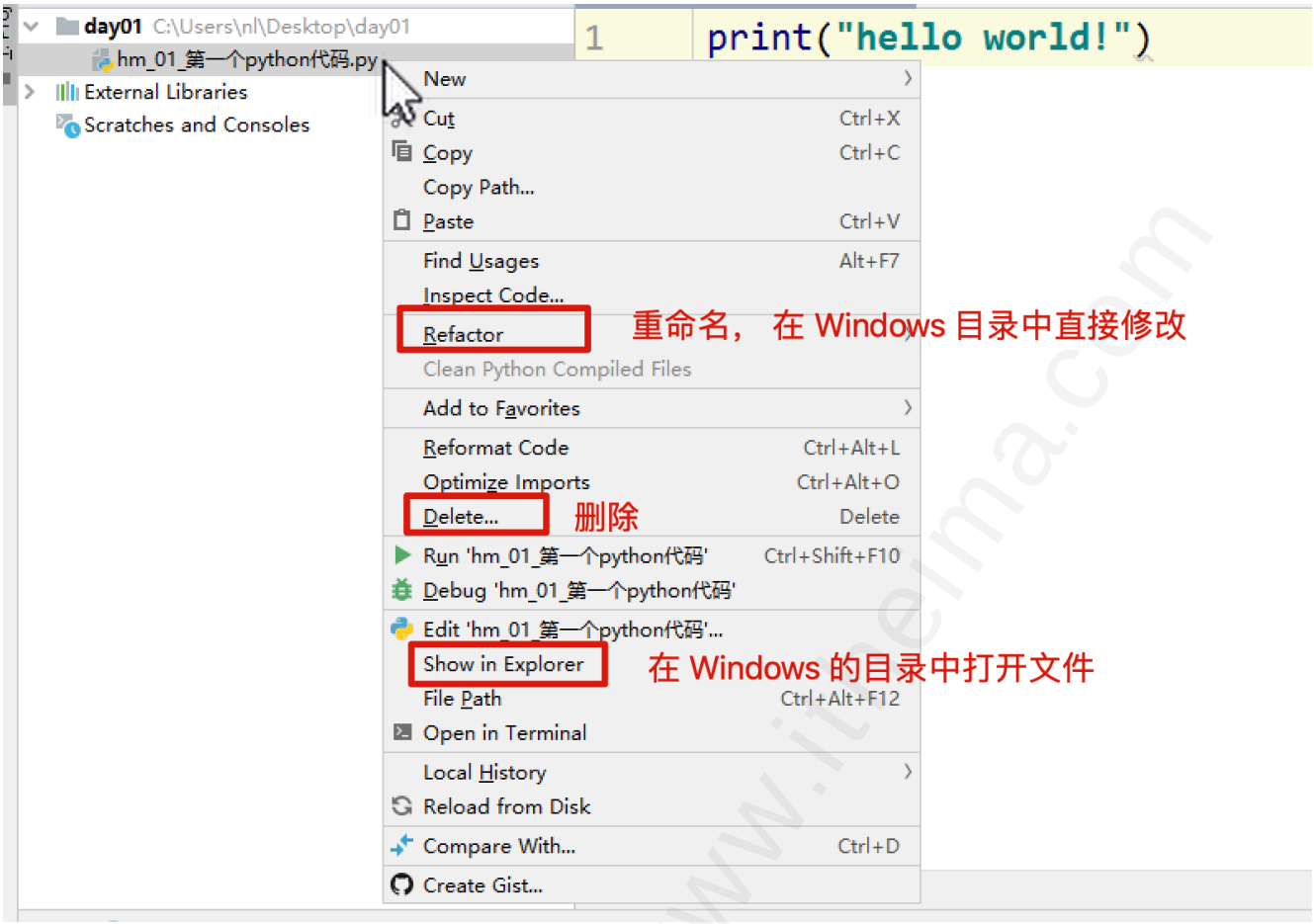
- 设置背景色



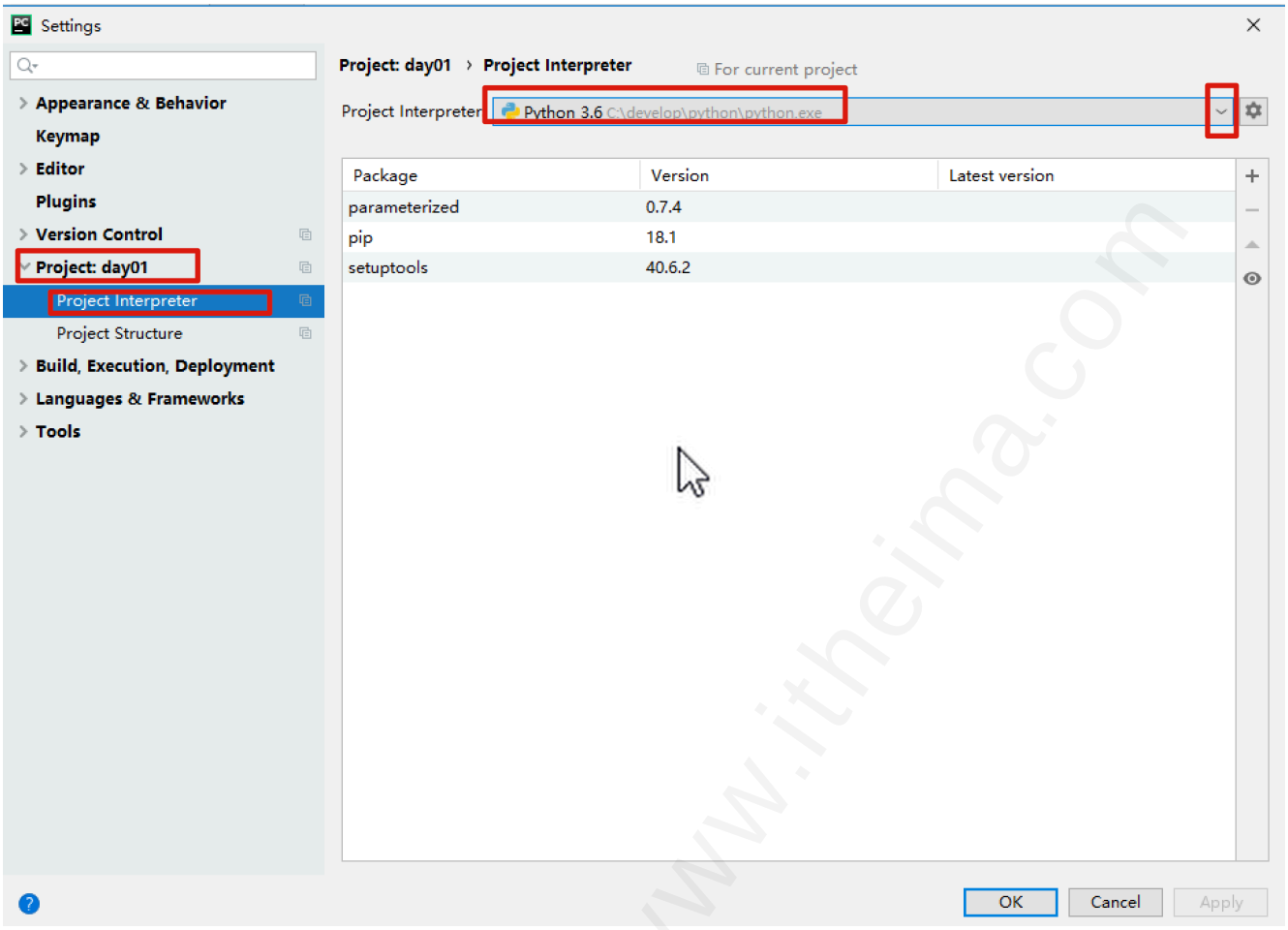
- 设置代码的字体和大小

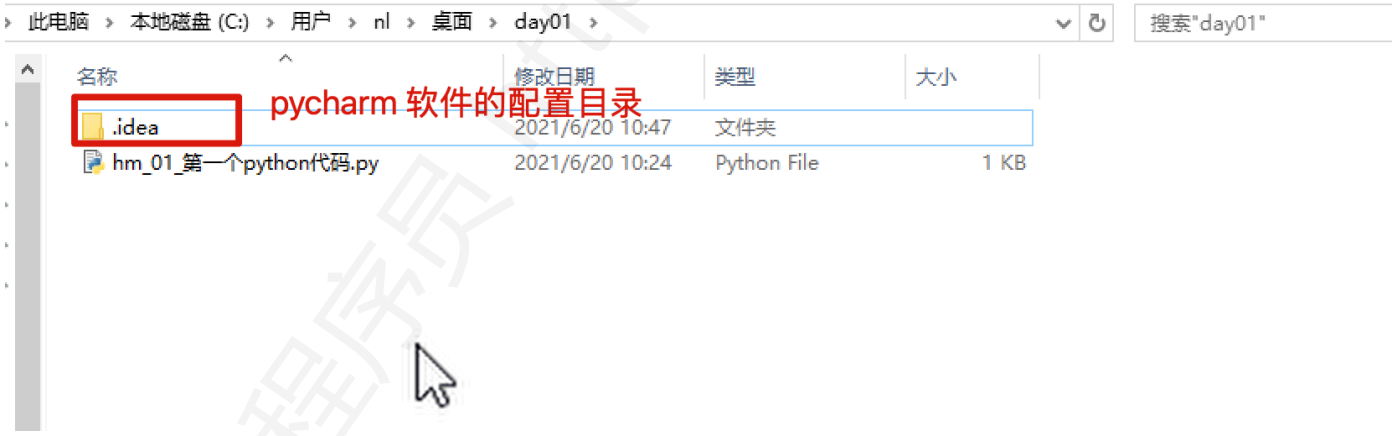


- 右键菜单的使用



- 设置解释器





print 函数的简单使用

```
print("hello world!")
```

print() 是 Python 中自带的函数,作用在控制台中输出 括号中的内容
后续看到这个函数就是输出打印 数据的, 或者想要在控制台中显示某个内容,就要使用 print() 函数

print() 主要在学习阶段使用, 便于我们确认结果的正确性
在实际工作的代码中,基本不会使用 print,会使用 其他的内容代替(日志模块)

print() 函数中是什么内容,就会显示什么内容, 里边的文字信息 可以使用单引号,也可以使用 双引号

注释

1. 注释是对代码解释说明的文字, 不会执行, 可以增加代码的可读性
2. Python 中的注释分为两种, 单行注释和多行注释

- 单行注释

使用 井号空格进行注释(单独一个# 也可以)

快捷键 Ctrl(cmd) /
1. 可以选中多行,使用快捷键
2. 如果代码已经添加注释, 再次使用快捷键,会取消注释

- 多行注释

多行注释中的内容 可以换行书写

多行注释可以使用 3 对 双引号或者 3 对 单引号 , 被三对引号包括的内容就是注释的内容

三对引号的注释,一般写在文件的最开始部分,或者文档注释处(函数)

```
#这是单行注释,代码不会执行
# 以井号空格开始的注释
print('hello world')
"""
使用三对双引号包括起来的内容 ,也是注释,
可以换行, 不会执行
"""
'''
使用三对单引号包括起来的内容 ,也是注释,
可以换行, 不会执行
'''
print('end')
```

Python 代码中三种波浪线和 PEP8

- 红色

  红色波浪线是代码的错误, 必须处理,代码才能执行

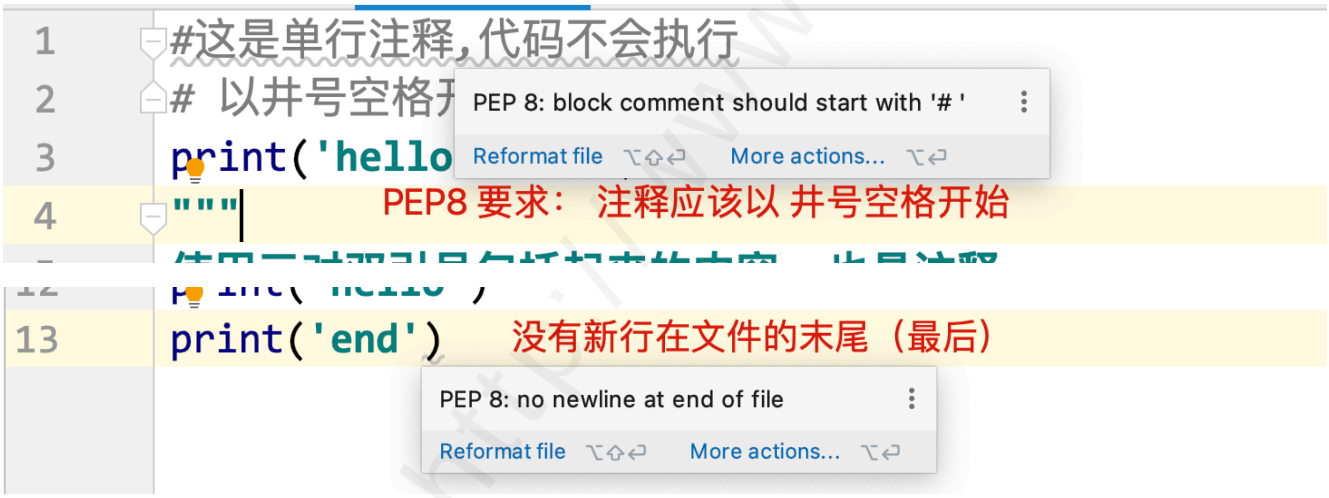
  注意: 在后续课程中,某些代码没有写完,也会出现红色波浪线

- 灰色

灰色波浪线，不会影响代码的正常执行，基本上所有的灰色波浪线都是 PEP8 造成的

PEP8：是 Python 代码的书写规范，如果不按照这个规范书写，会给灰色波浪线提示，建议代码的书写按照 PEP8 的规范书写

1. 可以书写代码的时候注意 PEP8 的代码规范
2. 可以在书写完成之后，使用快捷键 Ctrl Alt L 来按照 PEP8 的规范自动格式化代码



- 绿色

绿色波浪线，不影响代码的正常执行，在引号中，认为你书写的内容不是一个单词，就会给你绿色提示.

在 cmd 终端中运行 Python 代码

python 代码文件的名字

```
C:\Users\nl>python C:\Users\nl\Desktop\day01\hm_01_第一个python代码.py
hello world!

C:\Users\nl>
```

```
C:\Users\nl\Desktop\day01>dir
 驱动器 C 中的卷没有标签。
 卷的序列号是 FEDB-DA1E

 C:\Users\nl\Desktop\day01 的目录

2021/06/20  11:29    <DIR>          .
2021/06/20  11:29    <DIR>          ..
2021/06/20  14:04    <DIR>          .idea
2021/06/20  11:29                30 hm_01_第一个python代码.py
               1 个文件             30 字节
               3 个目录 16,866,168,832 可用字节

C:\Users\nl\Desktop\day01>python hm_01_第一个python代码.py
hello world!

C:\Users\nl\Desktop\day01>
```

```
(py36) ➜  04-代码 python hm_01_第一个python代码.py
hello world!
hello python!
(py36) ➜  04-代码
```

也是一个 cmd 窗口

Terminal | Python Console | 4: Run | 6: TODO

变量

变量

作用：是用来存储数据的(在程序代码中出现的数据,想要保存下来使用，就必须使用变量)，如：测试数据，用户名，密码，验证码

变量注意事项：变量必须先定义(保存数据)后使用(取出数据).

定义变量

```
变量名 = 数据值    # 可以理解为 是将数据值保存到变量中
# 比如:
name = '张三' # 定义一个变量 name, 存储的数据值是 张三
```

使用变量

变量定义之后，想要是使用变量中的数据，直接使用变量名即可

```
# 使用变量获取数据，打印
print(name)
```

```python
# 定义一个变量,存储你的名字
name = '张三'

# 使用变量打印名字，不需要加引号
print(name)    # 张三
# 如果给 name 添加引号，添加引号之后,输出的就是引号中的内容
print('name')  # name
```

变量名的命名规范

起名字的规范，标识符的规则

1. 必须由字母 数字和下划线组成, 并且不能以数字开头
2. 不能使用 Python 中的关键字作为变量名

   关键字: Python 自带的已经使用的标识符,具有特殊的作用
3. 区分大小写
4. 建议性的命名
   - 驼峰命名法
     - 大驼峰: 每个单词的首字母大写 `MyName`
     - 小驼峰: 第一个单词的首字母小写,其余单词的首字母大写 `myName`

- 下划线连接法: 每个单词之间使用下划线连接 `my_name`

> Python 中的变量的定义使用的是 下划线连接

- 见名知意

> name 姓名 age 年龄 height 身高 ....

数据类型

> 将生活常见的数据划分为不同的类型，因为不同的类型可以进行的操作是不一样的，数字需要加减乘除 ，文字不需要 ...

- 数字类型
  - 整型 (`int`) , 就是整数 , 即不带小数点的数
  - 浮点型(`float`), 就是小数
  - 布尔类型(`bool`), 只有两个值
    - 真 `True` , `1`
    - 假 `False` `0`, 非 0 即真

    > True 和 False 都是 Python 中的关键字，注意大小写,不要写错了

  - 复数类型 `3 + 4i`, 不会用的

- 非数字类型
  - 字符串: (`str`) 使用引号引起来的就是字符串
  - 列表 (`list`) `[1, 2, 3, 4]`
  - 元组(`tuple`) `(1, 2, 4, 4)`
  - 字典 (`dict`) `{'name': '小明', 'age': 18}`

- type() 函数

```
可以获取变量的数据类型
type(变量)

想要将这个变量的类型在控制台显示，需要使用 print 输出
print(type(变量))
```

```
# 整型  <class 'int'>
age = 18
print(type(age))   # type(age).print 回车

# 浮点型  <class 'float'>
height = 1.71
print(type(height))
# 布尔类型  <class 'bool'>  True False
isMen = True
print(type(isMen))
# 字符串类型，使用引号引起来的就是字符串  <class 'str'>
name = '小明'
```

```
print(type(name))

num = '18'
print(type(num))  # str

num = 'True'
print(type(num))   # str
```

类型转换

根据代码的需要，将一种数据类型转换另一种数据类型(将 input 输入得到的数字转换为整型)

语法:

变量 = 要转换为的类型(原数据)

1. 数据原来是什么类型
2. 你要转换为什么类型

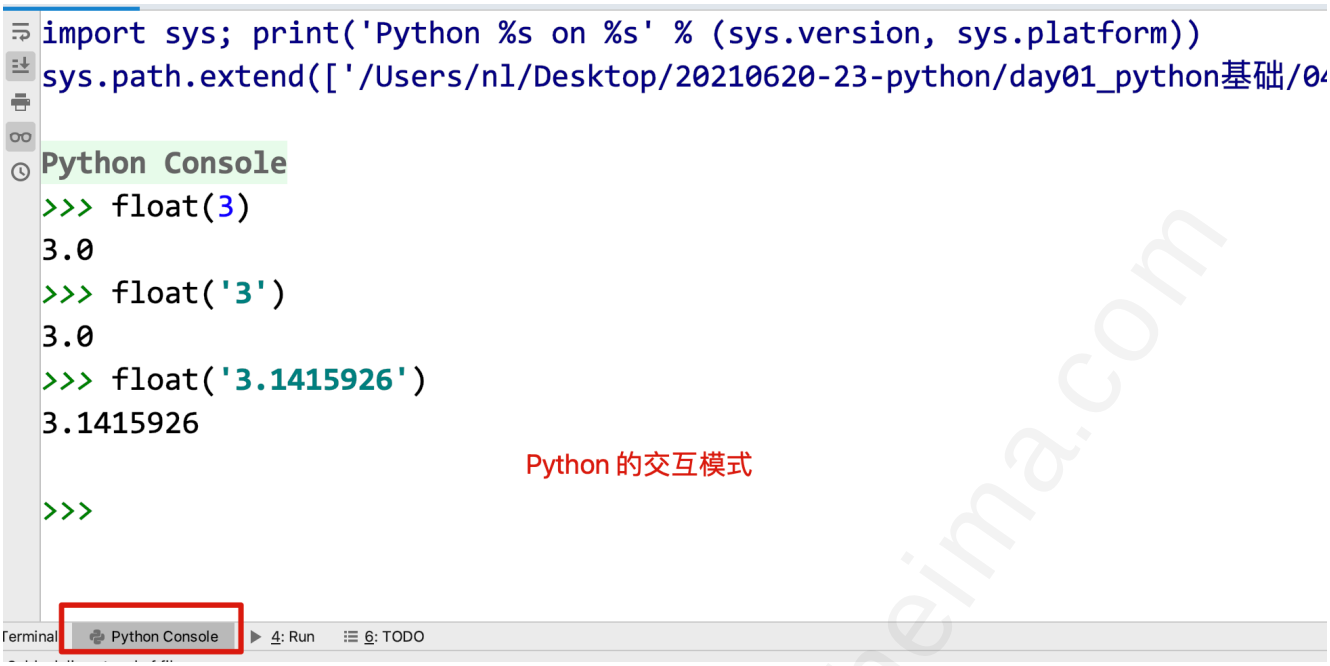
注意点：数据类型转换,不会改变原来的数据的类型，会生成一个新的数据类型

```python
# 将 input 输入得到的数字转换为整型
age = input('请输入你的年龄:')

print('age 本来的类型 :', type(age))

# 类型转换
age1 = int(age)
print('转换后 age的类型 :', type(age))
print('转换后 age1的类型:', type(age1))
```

```
请输入你的年龄:18
age 本来的类型 : <class 'str'>
转换后 age的类型 : <class 'str'>
转换后 age1的类型: <class 'int'>
```

- `int()` 将其他类型转换为 int 类型
  1. 可以将 `float` 类型的数字转换为 整型
  2. 可以将 `整数类型的字符串` 转换为 整型 `3` `123`
- `float()` 将其他类型转换为 浮点型
  1. 可以将 `int` 类型转换为 浮点型 `float(3)  ---> 3.0`
  2. 可以将 `数字类型的字符串(整数类型和小数类型)` 转换为 浮点型
- `str()` 将其他类型转换为 字符串类型

> 任何类型都可以使用 `str()` 将其转换为字符串，一般都是直接加上引号



Python 的交互模式

输入

获取用户使用键盘录入的内容

使用的函数是 input()

变量 = input('提示的信息')

1. 代码从上到下执行，遇到 input 函数之后,会暂停执行,等待用户的输入，r如果不输入会一直等待
2. 在输入的过程中,遇到回车,代表本次输入结束
3. 会将你输入的内容 保存到等号左边的变量中，并且 变量的数据类型 一定是 str

```
result = input('请输入内容:')
print(type(result), result)  # 打印数据类型和 数据值

# 1. 直接回车   <class 'str'>
# 2. 小明   <class 'str'> 小明
# 3. 18 <class 'str'> 18
# 4. True   <class 'str'> True
```

输出

输出使用的函数是 print() 函数， 作用,将程序中的数据或者结果打印到控制台(屏幕)

```
print('hello world')
name = '小明'
print(name)
age = 18
print(name, age)   # 可以使用逗号输出多个内容
```

格式化输出

在字符串中指定的位置,输出变量中存储的值.

1. 在需要使用变量的地方,使用特殊符号占位
2. 使用变量填充占位的数据

- % 格式化输出占位符号
  - `%d` 占位, 填充 整型数据 `digit`
  - `%f` 占位. 填充 浮点型数据 `float`
  - `%s` 占位, 填充 字符串数据 `string`

    补充: 其实 %s 的占位符,可以填充任意类型的数据

```
# 定义变量 姓名  年龄  身高
name = '小明'  # 可以使用 input 输入
age = 18   # 可以使用 input 输入
height = 1.71  # 可以使用 input 输入

# 要求按照以下个数输出个人信息
# 我的名字是 xx, 年龄是 xx, 身高是 xx m
# 使用格式化输出实现
# print('我的名字是 name, 年龄是 age, 身高是 height m')
print('我的名字是 %s, 年龄是 %d, 身高是 %f m' % (name, age, height))
# 小数默认显示 6 位, 如果想要指定显示小数点后几位, %.nf , n 需要换成具体的整数数字,即保留小数的位置
print('我的名字是 %s, 年龄是 %d, 身高是 %.2f m' % (name, age, height))  # 两位小数
```

```
print('我的名字是 %s, 年龄是 %d, 身高是 %.1f m' %
(name, age, height))  # 一位小数

# 补充
stu_num = 1   # 学号
# 我的学号是 000001
print('我的学号是%d' % stu_num)
# %0nd n 需要换成具体的整数数字, 表示整数一共占几位
print('我的学号是%06d' % stu_num)

num = 90  # 考试的及格率
# 某次考试的及格率为 90%, 如果在 格式化中需要显示%,
在书写的使用 需要使用 两个 %% 才可以
print('某次考试的及格率为 %d%%' % num)
```

- F-string(f字符串的格式化方法)

```
f-string 格式化的方法,想要使用 ,Python 的版本 >= 3.6

1. 需要在字符串的前边加上 f"" 或者 F""
2. 占位符号统一变为 {}
3. 需要填充的变量 写在 {} 中
```

```
# 定义变量 姓名  年龄  身高
name = '小明'  # 可以使用 input 输入
age = 18   # 可以使用 input 输入
height = 1.71  # 可以使用 input 输入
```

```
stu_num = 1  # 学号
num = 90  # 及格率

# print('我的名字是 xx，年龄是 xx，身高是 xx m，学号 xx，本次考试的及格率为 xx%')
print(f'我的名字是 {name}，年龄是 {age}，身高是 {height} m，学号 {stu_num}，本次考试的及格率为 {num}%')
# 一般不会有这样的需求
print(f'我的名字是 {name}，年龄是 {age}，身高是 {height:.3f} m，学号 {stu_num:06d}，本次考试的及格率为 {num}%')

# 在字符串中想要输出换行 \n (转义字符)
print(f'我的名字是 {name}，年龄是 {age}，身高是 {height:.3f} m，学号 {stu_num:06d},\n本次考试的及格率为 {num}%')
```

快捷键(小操作)

添加引号括号：可以直接选中要添加引号或者括号的内容，书写即可

撤销 ： Ctrl Z
删除一行： Ctrl x
复制粘贴一行： Ctrl d

快速 在代码下方，新建一行： shift 回车

运算符

算术运算符

场景构建：假设有x y 二个变量，x=10，y=20

运算符	描述	实例
+	加	x + y = 30
-	减	x - y = -10
*	乘	x * y = 200
/	除	x / y = 0.5 得到的是浮点类型 10 / 2 = 5.0
//	求商 整数	x // y = 0（商为0）
%	求余 整数	x % y = 10
**	幂运算，指数运算	2 ** 3 = 2 * 2 * 2 = 8

被除数 ÷ 除数 = 商 ... 余数

优先级：先算谁,再算谁（不确定优先级,就使用 ()）
() > ** > * / // % > + -

```
>>> int(3.67)
3
>>> 1+2
3
>>> 8 / 2
4.0
>>> 9 / 2
4.5
>>> 9 // 2  # 求商
4
>>> 9  % 2
1
```

比较运算符

比较运算符得到都是 bool 类型

> < >= <=

== 判断两个数是否相等，相等为 True，不相等为 False

!= 判断两个数是否不相等，不相等为 True，相等为 False

```
>>> a = 5
>>> b = 3
>>> a > b
True
>>> a < b
False
>>> a == b
False
>>> a != b
True
>>> a >= b  # 5 >= 3
True
>>> a <= 5
True
```