

Table of Contents

软件测试Python课程	1.1
Python基础	1.2
认识Python	1.2.1
Python环境搭建	1.2.2
PyCharm	1.2.3
注释	1.2.4
变量	1.2.5
变量的类型	1.2.6
程序的输入和输出	1.2.7
运算符	1.2.8

软件测试Python课程

本阶段课程不仅可以帮助我们进入Python语言世界，同时也是后续UI自动化测试、接口自动化测试等课程阶段的语言基础。

Life is short, you need Python! -- 人生苦短，我用Python！

课程大纲

序号	章节	知识点
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

课程目标

1. 掌握如何搭建Python开发环境；
2. 掌握Python基础语法, 具备基础的编程能力；
3. 建立编程思维以及面向对象程序设计思想；
4. 掌握如何通过UnitTest编写测试脚本，并生成HTML测试报告。

佐智播客-黑马程序员

Python基础

目标

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

认识Python

目标

1. 了解 python 语言及其应用领域
2. 了解 Python 运行原理

1. Python介绍

由吉多·范罗苏姆(Guido van Rossum) 于1989年创造

1.1 为什么学习Python

- 简单, 易学, 免费, 开源, 适用人群广泛
 - 零基础学习
 - 跨行业转型
 - 运维人员
 - Web全栈开发
 - 测试人员
- 应用领域广泛
 - 自动化测试
 - 网络爬虫
 - Web 开发
 - 自动化运维
 - 数据分析
 - 人工智能
 - 机器学习

1.2 Python发展历史

- Python2.X
- Python3.X (主讲版本)
 - Python3.5
 - Python3.6
 - Python3.7
- 详细了解: [参考](#)

2. 语言分类简介

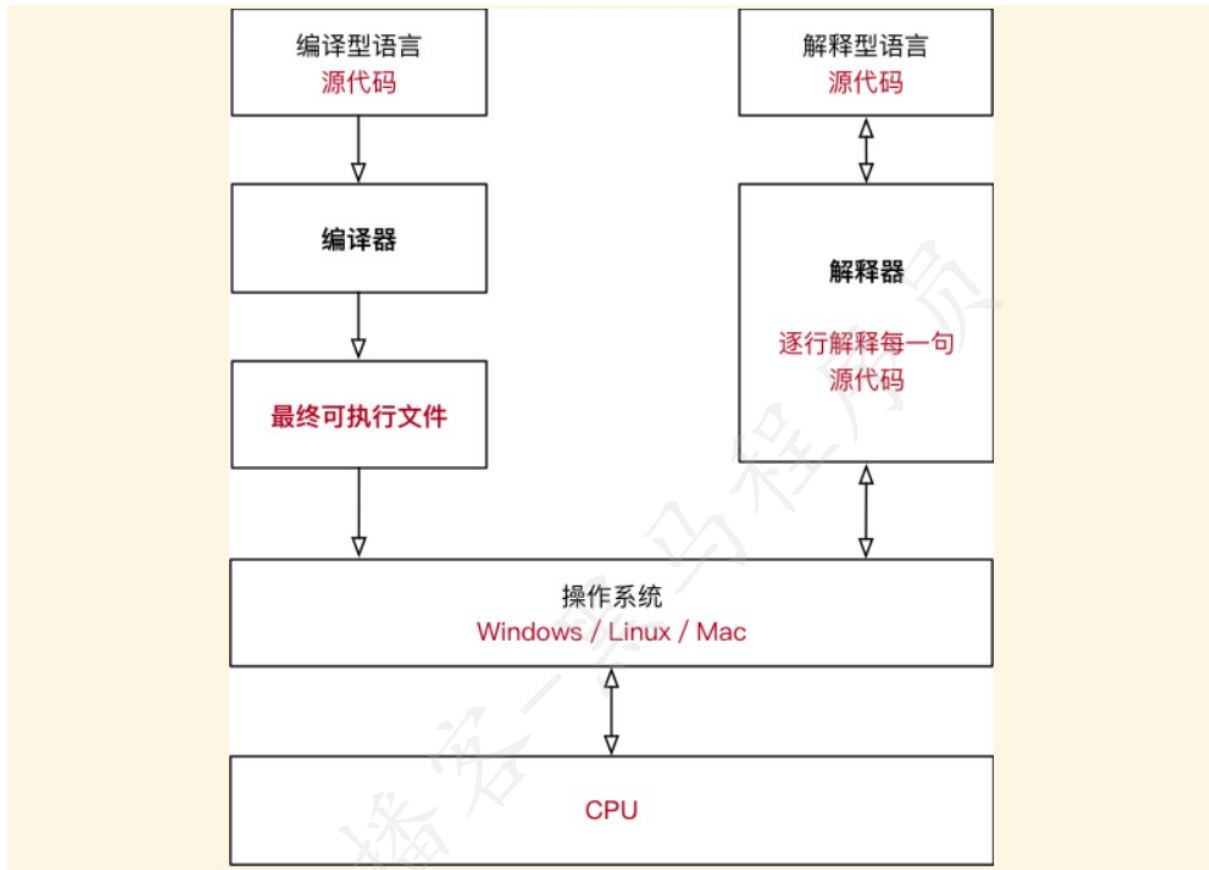
2.1 编译型

- 把程序源代码都编译成机器语言(二进制), 保存为二进制文件
- 计算机可以直接运行, 执行速度快
- C, C++, GO, Swift, Object-C ...

2.2 解释型

- 在程序执行的时候才会一行一行的处理成机器语言
- 执行速度慢
- JavaScript, Python, Ruby, PHP...

本质：计算机本身只能识别机器语言(二进制)



Python环境搭建

目标

1. 掌握如何安装Python解释器
2. 掌握如何使用解释器运行Python脚本

1. Python 解释器

Python 是一门解释型语言，通过解释器来运行 .py 文件

1.1 Python解释器下载

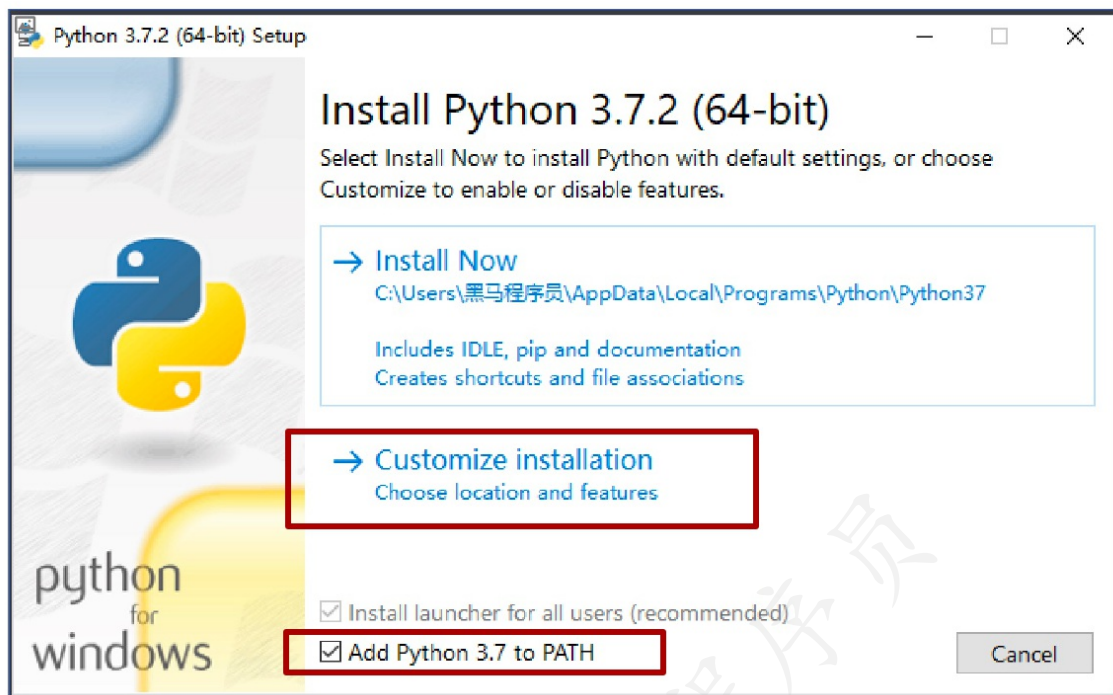
1. 下载对应的解释器安装包，官方解释器：<https://www.python.org/downloads/release>
2. 选择对应的版本：建议使用3.5之后版本
3. 点击目标版本，进入指定下载界面
4. 下拉最后，选择对应平台适配版本

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		9a080a86e1a8d85e45eee4b1cd0a18a2	22930752	SIG
XZ compressed source tarball	Source release		c3f30a0aff425dda77d19e02f420d6ba	17156744	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	c58267cab96fd291d332a2b163edd33	28060853	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	3ad13cc51c488182ed21a50050a38ba7	26954940	SIG
Windows help file	Windows		e01b52e24494611121b4a866932b4123	8139973	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	7148ec14edfdc13f42e06a14d617c921	7186734	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	767db14ed07b245e24e10785f9d28e29	31930528	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	f30be4659721a0ef68e29cae099fed6f	1319992	SIG
Windows x86 embeddable zip file	Windows		b4c424de065bad238c71359f3cd71ef2	6401894	SIG
Windows x86 executable installer	Windows		467161f1e894254096f9a69e2db3302c	30878752	SIG
Windows x86 web-based installer	Windows		a940f770b4bc617ab4a308ff1e27abd6	1293456	SIG

1.2 Python解释器安装

1. 双击安装，执行下一步安装
2. 选择自定义安装



3. 等待 完成安装
4. 验证是否成功: cmd进入命令行, 输入python 查看信息

```
C:\Users\yajiru\Desktop
λ python
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 13:35:33) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> exit()
```

1. 输入python 命令
2. 回车执行
3. 使用 exit() 退出

注意:

1. 默认安装路径可以使用, 但是不方便将来查找第三方包存放目录
2. 自定义目录建议直接放置于某个盘符根下, 类似: C:/Python36 或 D:/python37...
3. 强烈建议勾选将路径添加至 Path

2. 解释器运行Python脚本

前置条件: Python 解释器安装完毕

2.1 演练步骤

1. 在桌面新建测试目录 demo
2. 在 demo 目录下新建 demo.py 文件
3. 使用 Notepad++ 工具打开 demo.py, 输入 print("第一个python程序")

4. 在 demo 目录下进入到 cmd 命令模式

5. 使用 python demo.py 运行程序

注意:

- 1.在 windows 上新建的文件，有些操作系统默认编码是 gbk
- 2.建议直接使用 notepad++ 新建文件，保存为demo.py

PyCharm

目标

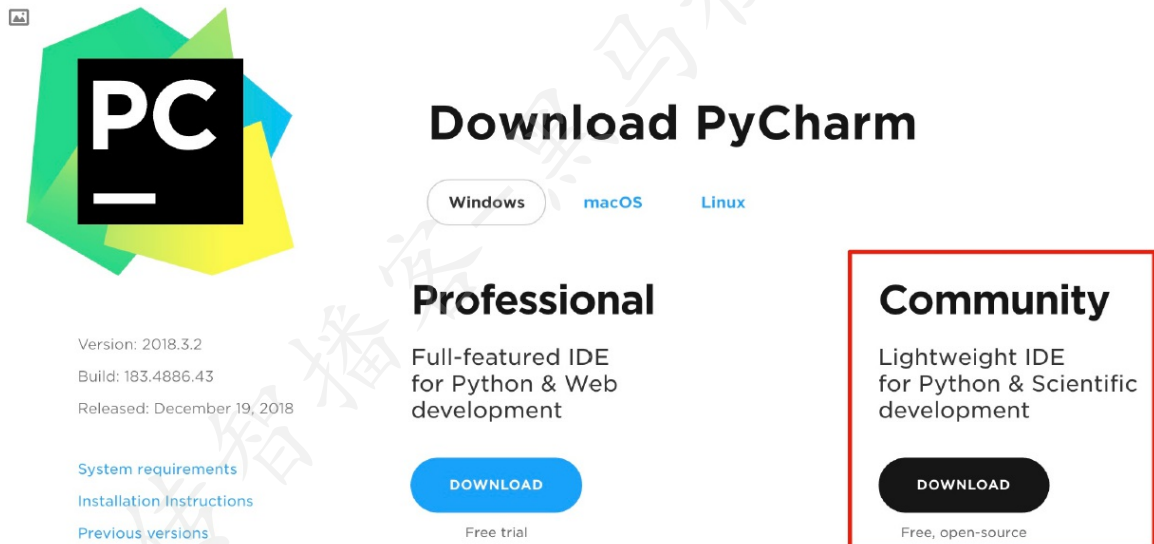
1. 知道如何安装PyCharm
2. 掌握如何在PyCharm中编写Python代码并运行

1. PyCharm介绍

- Pycharm 是一种 Python IDE (集成开发环境), 内置功能丰富提高开发效率
- Pycharm 分为专业版(professional) 和 社区版(community), 这里使用社区版

2. PyCharm安装

1. 下载安装包, 下载地址: <http://www.jetbrains.com/pycharm/download>



Version: 2018.3.2
Build: 183.4886.43
Released: December 19, 2018

[System requirements](#)
[Installation Instructions](#)
[Previous versions](#)

Download PyCharm

Windows macOS Linux

Professional

Full-featured IDE for Python & Web development

DOWNLOAD

Free trial

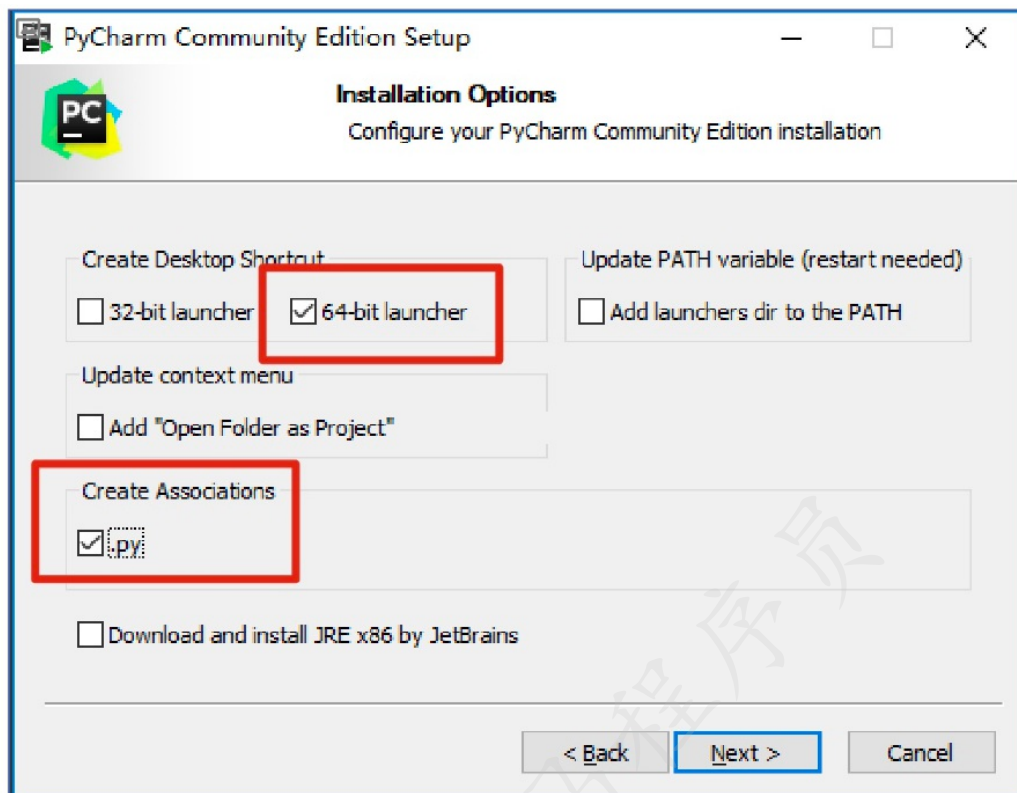
Community

Lightweight IDE for Python & Scientific development

DOWNLOAD

Free, open-source

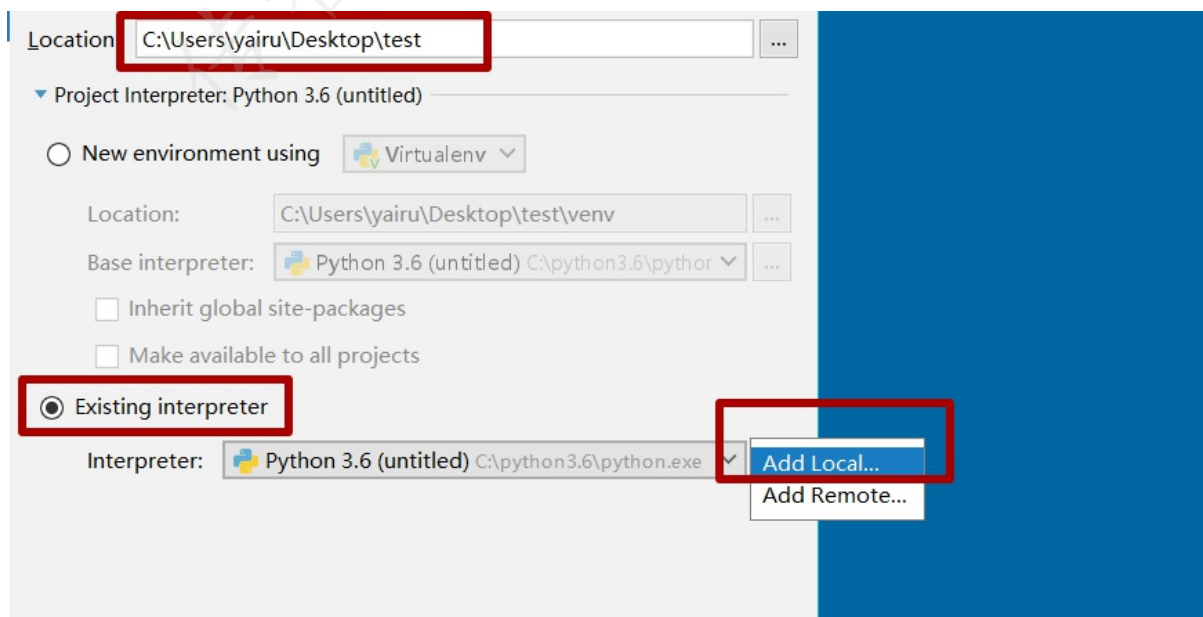
2. 执行安装程序



3. PyCharm基本使用

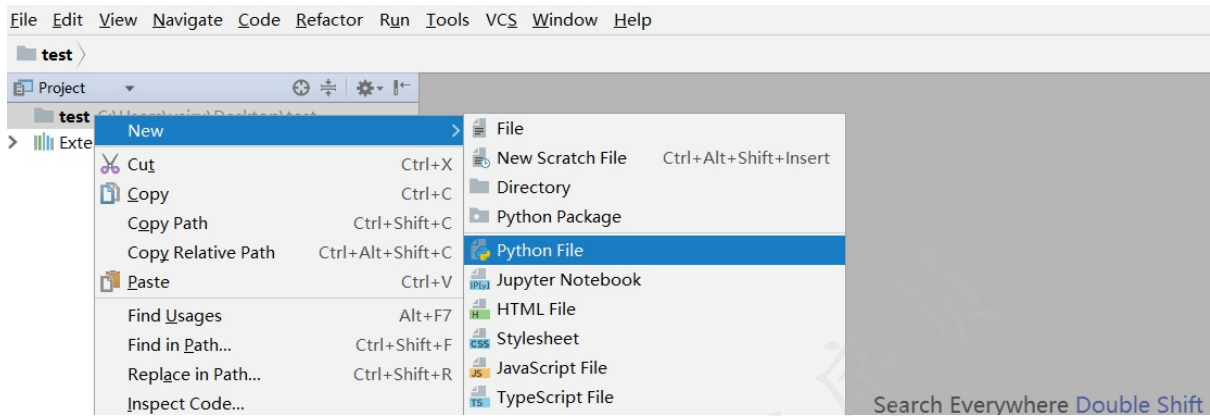
3.1 新建项目

1. 打开 Pycharm
2. 选择 [Create New Project]
3. 指定本地的 Python 解释器
4. 点击 create 创建完成



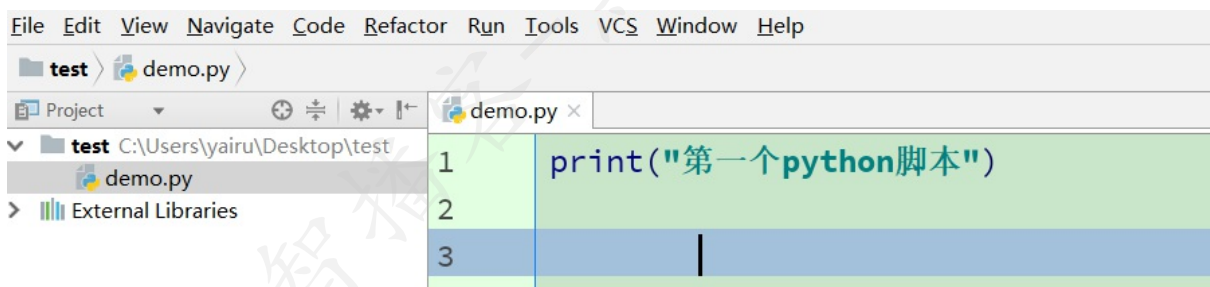
3.2 新建文件

1. 进入 Pycharm 项目
2. 选中对应项目名，鼠标右击
3. 点击 New 选择 Python file
4. 自定义文件名然后选择创建



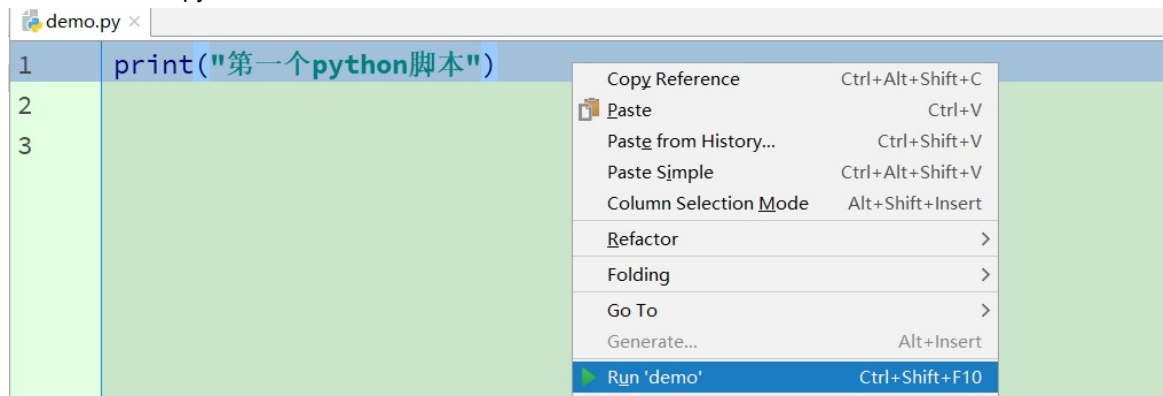
3.3 编写代码运行

- 按语法书写 python 脚本
- 书写 `print("第一个python脚本")`
- 在工具栏或直接运行 py 脚本

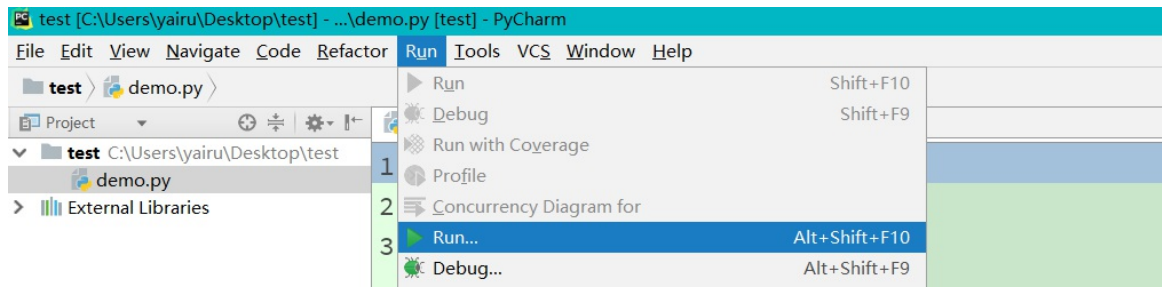


3.4 查看运行结果

- 方式一：在当前 py 文件编辑区直接鼠标右击 选择Run



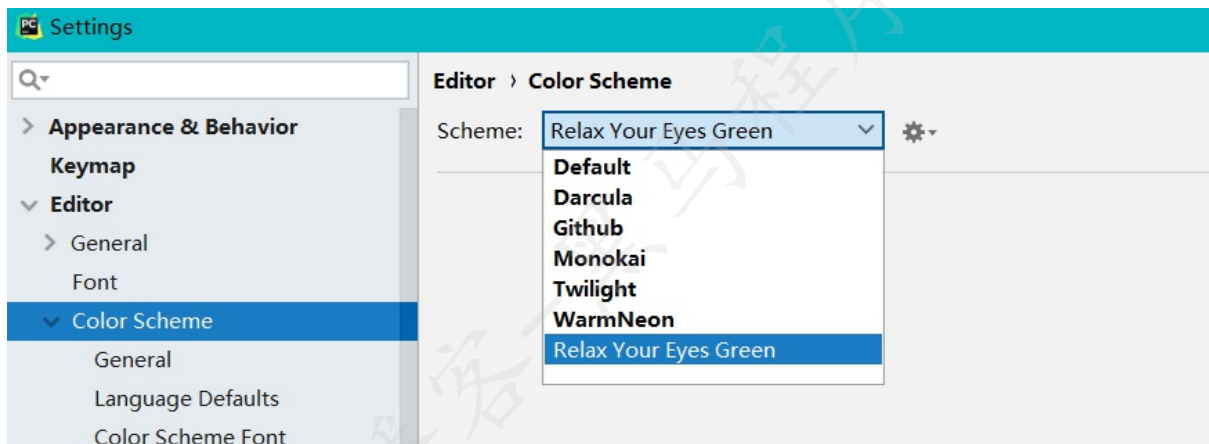
- 方式二：打开对应文件，在工具栏中选择 Run



4. PyCharm常用配置

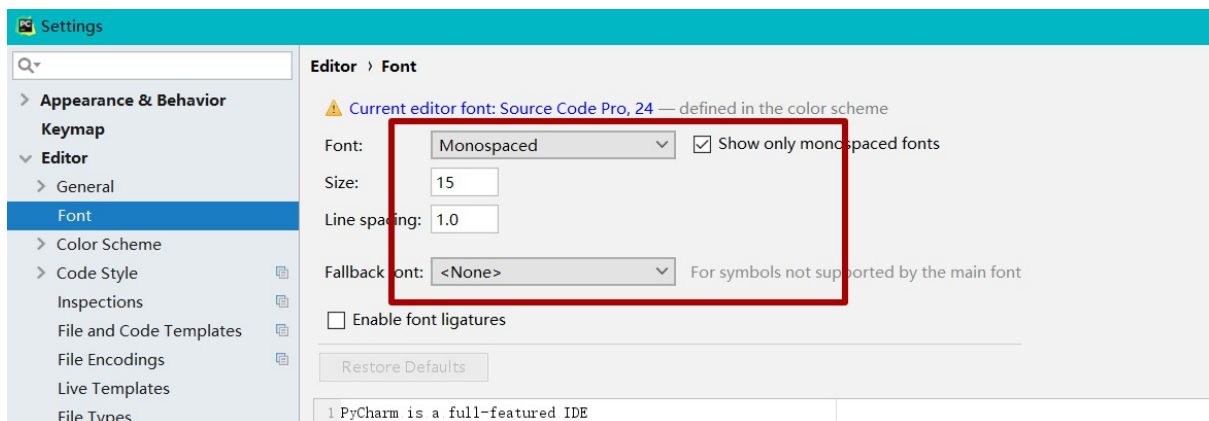
4.1 修改主题配置

1. 点击菜单 File(文件), 选择 Settings(设置) 选项
2. 选择 Editor(编辑), 点击 Color Scheme(配色方案)
3. 在右侧选择对应的主题配置



4.2 自定义字体设置

1. 点击菜单 File(文件), 选择 Settings(设置) 选项
2. 选择 Editor(编辑)
3. 选择 Font(字体)
4. 右侧设置具体的字体名称、大小、行高



佐智播客-黑马程序员

注释

目标

1. 掌握单行注释和多行注释的使用方式

1. 注释的作用

- 使用开发者自己熟悉的语言在程序代码中添加标注或说明
- 通过注释增强程序代码可读性
- 注释的内容不会被解释器执行

未添加注释代码:

```
import allure
import pytest
import Base
import Handle

class TestLogin:
    def setup(self):
        self.driver = Base.get_driver()
        self.handle = Handle.TotalHandle(self.driver)
        self.handle.init_home.tap_me()
        if not self.handle.init_me.assert_goon_exist():
            self.handle.init_me.tap_set()
            self.handle.init_set.tap_quit()
            self.handle.init_set.tap_confirm()
            self.handle.init_home.tap_me()

        self.handle.init_me.tap_logon()

@pytest.mark.parametrize("info", Base.get_data("login", "login_miss_num"))
def test_login_miss_num(self, info):
    self.handle.init_login.input_num(info["num"])
    self.handle.init_login.input_pwd(info["pwd"])
    self.handle.init_login.tap_login()
    assert self.handle.init_login.assert_toast("此用户不存在")
```

已添加注释代码:

```
import allure
import pytest
import Base
import Handle

# 登录功能点测试脚本
class TestLogin:

    # 前置步骤
    def setup(self):
        self.driver = Base.get_driver()
        self.handle = Handle.TotalHandle(self.driver)
```

```

# 执行点击“我”动作
self.handle.init_home.tap_me()

# 执行是否登录的判断
if not self.handle.init_me.assert_goon_exist():
    # 步骤1: 点击设置
    self.handle.init_me.tap_set()

    # 步骤2: 滑屏找到退出, 点击
    self.handle.init_set.tap_quit()

    # 步骤3: 点击确认
    self.handle.init_set.tap_confirm()

    # 步骤4: 点击我
    self.handle.init_home.tap_me()

# 执行点击 去登录 动作
self.handle.init_me.tap_logon()

# 账号不存在测试场景
# 使用参数化实现数据驱动文件读取
@pytest.mark.parametrize("info", Base.get_data("login", "login_miss_num"))
def test_login_miss_num(self, info):

    # 步骤1: 输入账号
    self.handle.init_login.input_num(info["num"])

    # 步骤2: 输入密码
    self.handle.init_login.input_pwd(info["pwd"])

    # 步骤3: 点击登录
    self.handle.init_login.tap_login()

    # 步骤4: 断言
    assert self.handle.init_login.assert_toast("此用户不存在")

```

2. 注释分类

注释分为：单行注释 + 多行注释(块注释)

2.1 单行注释

```
# 一次只注释一行内容
```

2.2 多行注释(块注释)

1. 使用六个双引号，之间部分为注释内容 `""" 被注释内容 """`
2. 使用六个单引号，之间部分为注释内容 `'''被注释内容'''`

2.3 注释快捷键

ctrl + /

1. 在需要被注释的单行代码所在行任意处执行即可
2. 选中需要被注释的整段内容 执行即可
3. 添加注释和去除注释规则是一样的

3. 注释扩展

3.1 何时添加注释

- 注释并非越多越好，一目了然的代码无需注释
- 逻辑复杂的代码，应当先写注释再编码
- 不应用中文去翻译 `python` 代码的含义

3.2 代码规范

- `Python` 官方提供了一套 PEP(Python Enhancement Proposals) 文档
- 其中第8篇文档专门针对 `Python` 的代码格式给出了建议，俗称 PEP8
- 官方地址：<https://www.python.org/dev/peps/pep-0008/>

变量

目标

1. 掌握变量的使用

1. 变量作用

场景举例

超市购物时会将随身物品存放于储物柜中，然后得到一个“编号”。购物结束后，通过此编号就可以找到我们的物品

场景分析

- 超市购物就是一个真实环境下的业务需求
- 随身物品就是完成整个业务需要用到的“数据”
- 我们最终需要获取的是 随身物品，找到物品使用了编号
- 可以认为是编号帮我们存储了 随身物品

变量作用

1. 程序执行的过程都是发生在内存当中的
2. 内存中的数据都是临时存储的
3. 为了快速定位目标数据，我们将数据在内存空间占据的空间分配一个名称，即变量

总结：变量将运算的中间结果暂存到内存,便于程序后续调用

2. 变量定义

2.1 语法

变量名 = 数据值

- 变量名自定义, 要满足 标识符 命名规则
- 数据值为要存储的数据，可以为不同的数据类型
- 单等号用于 赋值，不具备比较功能

2.2 标识符规则

标识符命名规则是Python中定义各种变量名或其他名字的时候的统一规范, 具体如下:

- 由数字, 字母, 下划线组成
- 不能使用数字开头
- 不能使用Python内置关键字
- 严格区分大小写

- 不能使用中文定义变量名

Python中的关键字:

```
False    None    True    and    as    assert    break    class
continue def    del    elif    else    except    finally    for
from     global  if     import  in     is     lambda    nonlocal
not      or     pass   raise   return  try    while     with
yield
```

2.3 命名习惯

1. 见名知义
2. 使用驼峰体
 - 大驼峰: 每个单词首字母大写, 例如: `MyName`
 - 小驼峰: 第二个单词开始首字母大写, 例如: `myName`
3. 使用下划线: 每个单词之间使用下划线连接, 例如: `my_name`

3. 变量的使用

3.1 使用步骤

1. 按标识符规则定义变量
2. 使用单等号给变量赋值
3. 在程序中使用变量名指代具体数据值

3.2 示例代码

```
name = syy
age = 18
print(name)
print(age)
```

变量的类型

目标

1. 掌握常见的数据类型
2. 掌握如何检测数据的类型
3. 掌握常见的数据类型转换方法

1. 数据为什么需要类型

1. 人类可以肉眼观察轻松的区别不同类型数据，但是计算机做不到。
2. 计算机工作的过程就是完成不同类型的计算，例如做数学运算，做文件存储，做逻辑判断
3. 为了让计算机完成不同的运算过程，在Python中就定义了数据类型的概念
4. 数据类型可以对不同数据进行分类管理和标识

2. 常见数据类型分类

- 数字型
 - 整型
 - 浮点型
 - 布尔型
 - 复数型(了解)
- 非数字型
 - 字符串型
 - 列表
 - 元组
 - 字典

注意：

1. 这里只讨论 3.X之后Python版本常用类型
2. 布尔型中非0即为真

3. 数据类型检测

使用 `type()` 函数可以查看一个变量的类型

```
age = 20
print(type(age))
```

```
a = 1
print(type(a)) # <class 'int'> -- 整型

b = 1.1
print(type(b)) # <class 'float'> -- 浮点型
```

```

c = True
print(type(c)) # <class 'bool'> -- 布尔型

d = '12345'
print(type(d)) # <class 'str'> -- 字符串

e = [10, 20, 30]
print(type(e)) # <class 'list'> -- 列表

f = (10, 20, 30)
print(type(f)) # <class 'tuple'> -- 元组

h = {10, 20, 30}
print(type(h)) # <class 'set'> -- 集合

g = {'name': 'TOM', 'age': 20}
print(type(g)) # <class 'dict'> -- 字典

```

4. 数据类型转换

作用: 将一个中数据类型转换为另一种数据类型

函数	说明
<code>int(x)</code>	将x转换为一个整数
<code>float(x)</code>	将x转换为一个浮点数
<code>str(x)</code>	将 x 转换为字符串
<code>tuple(s)</code>	将序列 s 转换为一个元组
<code>list(s)</code>	将序列 s 转换为一个列表

注意:

- `int(x)`, x是字符串时,必须是数字类型的字符串
- `float(x)`, x是字符串时,必须是浮点数类型的字符串
- `str(x)`, 对于任意数据类型x都可以转为字符串类型

程序的输入和输出

目标

1. 掌握程序输出的基本用法
2. 掌握如何实现格式化输出
3. 掌握如何实现程序的输入

1. 程序的输出

1.1 基本输出

输出的作用: 将程序运行结果输出打印到屏幕或终端

```
print("hello Python")

age = 18
print(age)
```

1.2 格式化输出

所谓格式化输出就是按照一定的格式输出内容

```
name = "小明"
print("姓名是: %s" % name)
```

常见的格式化符号

格式符号	转换
%s	字符串
%d	有符号的十进制整数
%f	浮点数
%%	输出 %

技巧

- %06d, 表示输出的整数显示位数, 不足位以0补全, 超出当前位数则原样输出
- %.2f, 表示小数点后显示的小数位数, 此为保留2位小数

格式化字符串

格式化字符串除了%s, 还可以写为 f'{{表达式}}'

```
age = 18
name = 'TOM'
weight = 75.5
student_id = 1
present = 95
```

```
# 我的名字是TOM
print('我的名字是%s' % name)

# 我的学号是0001
print('我的学号是%d' % student_id)

# 我的体重是75.50公斤
print('我的体重是%.2f公斤' % weight)

# 合格率是95%
print('合格率是%d%' % present)

# 我的名字是TOM，今年18岁了
print('我的名字是%s，今年%d岁了' % (name, age))

# 我的名字是TOM，明年19岁了
print('我的名字是%s，明年%d岁了' % (name, age + 1))

# 我的名字是TOM，明年19岁了
print(f'我的名字是{name}，明年{age + 1}岁了')
```

f-格式化字符串是Python3.6中新增的格式化方法，该方法更简单易读

1.3 转义字符

- `\n` : 换行
- `\t` : tab键

思考：为什么两个print()会换行输出呢？

在Python中，print()函数默认自带 `end="\n"` 这个换行结束符，所以导致每两个 print 直接会换行展示，用户可以按需求更改结束符

```
print("输出的内容", end="\n")
```

2. 程序的输入

在Python中，程序用来接收用户输入的数据的功能即是输入。

2.1 输入的语法

```
input("提示信息")
```

2.2 输入的特点

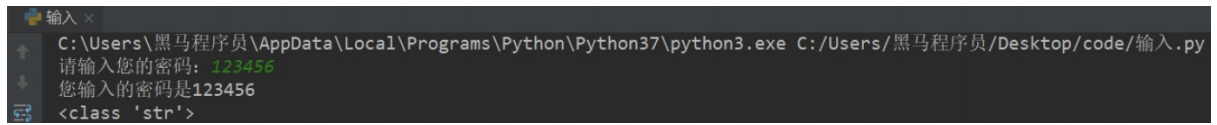
- 当程序执行到input，等待用户输入，输入完成之后程序才继续向下执行
- 在Python中，input 接收用户输入信息后，一般保存到变量中，方便使用
- 在Python中，input 会把用户输入的任意数据都当做字符串处理

```
password = input('请输入您的密码：')

print(f'您输入的密码是{password}')
```

```
print(type(password)) # <class 'str'>
```

控制台输出如下:



```
C:\Users\黑马程序员\AppData\Local\Programs\Python\Python37\python3.exe C:/Users/黑马程序员/Desktop/code/输入.py
请输入您的密码: 123456
您输入的密码是123456
<class 'str'>
```


运算符

目标

1. 掌握Python运算符的分类和使用

1. Python运算符介绍

1.1 运算符作用

1. 计算机可以完成的运算分为很多种，不仅仅限于 加减乘除
2. 不同类型的运算需要依赖不同的符号
3. 运算符可以让计算机认识并处理对应的计算逻辑

1.2 运算符分类

1. 算数运算符：完全常规数字运算
2. 比较运算符：完成二个值的大小比较运算
3. 逻辑运算符：常用业务之间的逻辑判断
4. 赋值运算符：将运算符右侧的值，赋值给左侧容器

1.3 运算符使用

1. 准备运算过程中需要的数据
2. 选择对应的运算符规则
3. 执行程序代码完成最终运算

```
a = 1
b = 2
print( a+b ) # 通过加法运算符完成运算
```

2. 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
//	求商	x // y = 0 (商为0)
%	求余	x % y = 10

**	幂	2 ** 3 = 8 (2的3次方)
----	---	----------------------

注:

python 算数运算符中的 * 可以用于字符串计算

输入: print("中"*10)

输出: 中中中中中中中中中中

3. 比较运算符

场景构建: 假设有a b 二个变量, a=10, b=20

运算符	描述	实例
==	检查运算操作数的值是否相等, 如果是表示成立, 返回 True	print(a==b) 返回 False
!=	检查运算操作数的值是否不相等, 如果是表示成立, 返回 True	print(a!=b)返回True
>	检查左侧操作数的值是否大于右侧, 如果是表示成立, 返回 True	print(a>b)返回False
<	检查左侧操作数的值是否小于右侧, 如果是表示成立, 返回 True	print(a<b)返回True
>=	检查左侧操作数的值是否 大于或等于右侧, 如果是, 表示条件成立, 返回True	print(a>=b)返回 False
<=	检查左侧操作数的值是否 小于或等于右侧, 如果是, 表示条件成立, 返回True	print(a<=b)返回True

4. 逻辑运算符

场景构建: 假设有a b 二个变量, a=True, b=True

运算符	描述	实例
and	只有a b 二者都为True, 才会返回True	print(a and b) 返回True
or	只要 a b有一个为True, 则会返回 True	print(a or b) 返回True
not	取反操作, 如果True, not之后为 False	print(not a) 返回False

5. 赋值运算符

场景构建: 假设有a, b二个变量, a=10, b=20

运算符	描述	实例
=	基础赋值运算	a = 10, print(a) 得10
+=	包含加法规则的赋值运算	a += 20, 等价于a = a +20, print(a) 得30
-=	包含减法规则的赋值运算	a -= 10, 等价于a = a - 10, print(a) 得 0
*=	包含乘法规则的赋值运算	a = 4, 等价于a = a * 4, print(a) 得40
/=	包含除法规则的赋值运算	a /= 2, 等价于a = a / 2, print(a) 得5.0

//=	包含求商规则的赋值运算	a //= 3, 等价于a = a // 3, print(a) 得3
%=	包含求余规则的赋值运算	a %= 5, 等价于a = a % 5, print(a) 得0
=	包含求幂规则的赋值运算	a= 2, 等价于a = a ** 2, print(a) 得100

注意:

1. = , 单等号在程序语言中不表示比较, 用于赋值
2. 复合赋值运算符之间没有空格, 连接在一起

6. 运算符优先级

有计算就会运算符, 不同的运算符默认具备不同的优先级

6.1 分类别单独比较

- 算术运算符【由高至低】
 - () 小括号分组最优先
 - ** 幂等
 - * // % 按书写顺序
 - + -
- 逻辑运算符【由高至低】
 - () 小括号分组, 优先级最高
 - not
 - and
 - or

6.2 按类型整体比较【由高至低】

运算符	描述
* *	幂
* / % //	乘 除 取余数 取商
+ -	加法 减法
<= >= < >	比较运算符
== !=	等于运算符
= %= /= //= -= += **=	赋值运算符
not and or	逻辑运算符