# 浩辰CAD Linux版 2024

# Python开发指南

苏州浩辰软件股份有限公司

苏州浩辰软件股份有限公司

# 目录

# 1. 简介

浩辰 CAD GRXSDK 是浩辰结合 GRX 和 Python 开发的一组编程接口，用于支持用户使用 python 拓展 GstarCAD 平台软件的功能，例如创建自定义命令，创建自定义图元和实体，访问和修改图纸的各类图形数据，实现自定义绘图和图纸分析算法等。

# 2. 系统要求

Python (>=3.5, <=3.10)。

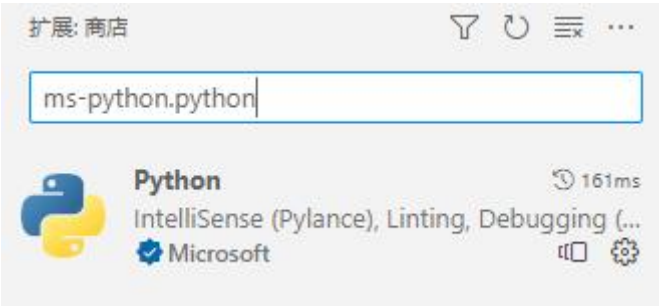
# 3. 配置开发环境

## 3.1 下载 vscode

访问 vscode 官网(https://code.visualstudio.com/Download#)，下载满足操作系统要求的安装包，按照 vscode 教程的说明，安装软件。

## 3.2 安装 python 插件

启动 vscode，在 vscode 商店搜索并安装拓展 ms-python.python，如下图。




## 3.3 配置 PYTHONPATH

启动 vscode，在 settings.json 文件中增加下面的配置。

```
{
"python.analysis.extraPaths": [
    "/opt/apps/gcstarcad/v2024/files/GcPyRuntime/Python3.5"],
"terminal.integrated.env.osx": {
```

```
"PYTHONPATH":
"/opt/apps/gcstarcad/v2024/files/GcPyRuntime/Python3.5"},
}
```

# 4. 开发样例

## 4.1 核心 module 简述

SDK 中包含 core 和 pygrx 两个子模块，通过下面的代码引用

```
from pygcad.core import *
from pygcad.pygrx import *
```

pygcad.core 中包含了@command 修饰符等 python 特有的核心接口；

pygcad.pygrx 中则包含与 Grx 接口一一对应的各种类型和方法；

## 4.2 使用@command 修饰符注册命令

在开发过程中，当一个用户自定义的函数被@command 修饰时，这个函数会在 Gcad 中被自动注册为命令，函数名即为默认的命令名。

例如下面的代码定义了一个名为`PYFUN`的命令：

```
@command()
def PyFun():
    pass
```

@commad 修饰符的声明如下：

```
def command(local_name=”“, global_name=”“, group_name=”“,
cmd_flags=0)
```

当 local_name 为空时，会自动使用被修饰的函数的名字作为 local_name；global_name 为空时，会自动使用 local_name 作为 global_name; group_name 为空时会选用 global_name 作为 group_name;

下面的代码定义了分别名为 PY_MY_CMDNAME_1(_2)的命令：

```
@command('PY_MY_CMDNAME_1')
```

```
def PyFun1():
    pass


@command(local_name='PY_MY_CMDNAME_2')
def PyFun2():
    pass
```

## 4.3 HelloPy.py 测试样例

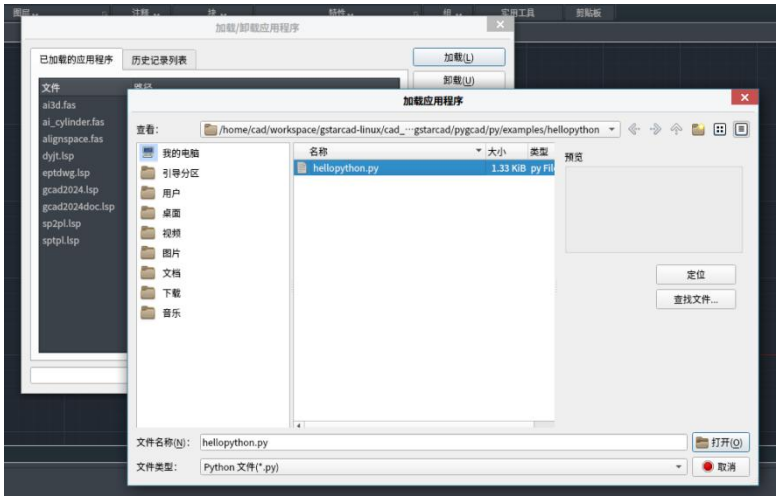
- 编写如下的 hellopy.py 文件

```
from pygcad.core.runtime import *
from pygcad.pygrx import *



@command()
def pyPrint():
    gcedPrompt('hellopython')
```

- 打开 gcad 软件，执行命令`APPLOAD`，在软件的提示下加载`hellopy.py`文件；




- 执行命令`PYPRINT`，可以看到命令提示符处输出“hellopython”；

## 4.4 调试 python 代码

**方法一：**使用 debugpy 进行调试(python>=3.6)

- 安装最新版本的 debugpy：

```
pip3 install debugpy
```

- 在 vscode 的 launch.json 文件中增加下面的内容：

```
{"name": "Python debugpy: Attach File",
 "type": "python",
 "request": "attach",
 "connect": {
"host": "127.0.0.1",
"port": 5678 }}
```

- 在准备调试 python 程序附近加入下面一段代码：

```
import debugpy
import sys
sys.executable='/usr/bin/python3' #如果 python 在其他路径，则修改成对应的路径
debugpy.listen(5678)
debugpy.wait_for_client()
debugpy.breakpoint()
```

- 启动 gcad，执行命令 APPLOAD 加载 python 程序，执行对应的命令，当程序执行到 debugpy.wait_for_client()时会自动阻塞，等待 vscode 的调试进程连接；
- 这时启动 vscode，选择 Python debugpy: Attach File 方式运行；
- 选择要附加的进程为 gcad；
- 这时 vscode 会正常进入断点，停在 debugpy.breakpoint()处；
- 除了可以用 debugpy.breakpoint()设置断点外，也可以从 vscode 的 UI 界面上直接设置断点，或者按 F9 设置断点(默认快捷键)；

### 方法 2：使用 snooper 进行调试(python=3.5)

由于 python3.5 存在兼容性问题，上述使用 debugpy 进行调试的方式已不可使用，我们建议使用 pysnooper 进行调试，具体方法如下：

- 安装 pysnooper：

```
pip3 install pysnooper
```

- 在想要调试的函数前添加`@snooper`修饰符：

```
import pysnooper


@pysnooper.snoop()
def func(x, y):
    z = x+y
    return z


res = func(1, 2)
print(res)
```

运行上面的代码之后，`pysnooper`会在控制台打印出下面的信息：

```
Source path:... test_snooper.py
Starting var:.. x = 1
Starting var:.. y = 2
17:12:24.080959 call         4 def func(x, y):
17:12:24.081087 line         5  z = x+y
New var:....... z = 3
17:12:24.081126 line         6  return z
17:12:24.081177 return       6  return z
Return value:.. 3
Elapsed time: 00:00:00.000300
```

- 如果只希望调试特定的代码块(而非整个函数体)，也可以使用下面的办法：

```
import pysnooper


def func():
    z = 0
    with pysnooper.snoop():
        for i in range(10):
            z += i
    return z


res = func()
print(res)
```

- 如果想要在 py 文件所在的目录下生成调试文件，可以为 snooper 指定 log 文件的输出目录，以及调试信息的关键字。

```
@pysnooper.snoop(output=gcutPysnooperDefaultOutput(__file__),
prefix='projectEllipse_debug: ')
def func():
    pass
```

## 5. 版权声明