Python & PDB Cheatsheet

如何开始调试

插入代码启动调试

```
import pdb; pdb.set_trace()
```

启动默认调试器pdb 不需要import pdb

```
breakpoint()
```

如不想修改代码 可修改运行指令

```
python -m pdb script.py
```

lineno是什么

lineno（行号）指源代码文件中的行号（从 1 开始）
在命令中使用时 它指定文件中的特定行

在当前文件的第 10 行设置断点

```
b 10
```

执行到当前函数的第 15 行

```
until 15
```

执行控制

执行当前行 进入被调用函数

```
s(tep)
```

执行到当前函数的下一行 不进入函数内部

```
n(ext)
```

继续运行直到断点

```
c(ont(inue))
```

执行到当前函数返回

```
r(eturn)
```

执行到指定行号或当前行之后
若无参数，执行完当前行（含循环）
指定行号指的是当前函数中的行号

```
unt(il) [lineno]
```

保留断点重启程序
重启时不会重新编译、加载模块 不会检测代码变更

```
run [args] / restart [args]
```

断点管理

设置断点 支持条件
无参数则会列出所有当前断点的详细信息

```
b(reak) [([filename:]lineno | function) [, condition]]
```

myfile.py 的第 20 行设置断点

```
b myfile.py:20
```

mymod/utils.py 的 add 函数入口设置断点

```
b mymod.utils.add
```

在第 10 行设置断点 当 x > 5 时触发

```
b 10 x > 5
```

在 add 函数入口设置断点 当 x == 10 时触发

```
b add x == 10
```

在 myfile.py 第 10 行设置条件断点（x > 5）

```
b myfile.py:10 x > 5
```

临时断点，命中一次后删除

```
tbreak [([filename:]lineno | function) [, condition]]
```

清除断点
无参数则会清除调试会话中设置的所有断点
不会修改或删除源代码中的 breakpoint()

```
cl(ear) [filename:lineno | bpnumber ...]
```

禁用断点（保留但不触发）

```
disable bpnumber ...
```

启用断点

```
enable bpnumber ...
```

设置断点忽略次数

```
ignore bpnumber [count]
```

为断点添加/删除条件

```
condition bpnumber [condition]
```

检查和显示

显示堆栈（箭头>表示当前帧）

```
w(here) [count]
```

在堆栈中上移动帧

```
u(p) [count]
```

在堆栈中下移动帧

```
d(own) [count]
```

列出源码（当前行周围 11 行 默认延续上次）

```
l(ist) [first[, last]]
```

列出当前函数全部源码

```
ll | longlist
```

显示当前函数参数

```
a(rgs)
```

打印表达式值

```
p expression
```

美观打印表达式

```
pp expression
```

查看当前帧的局部变量字典

```
p locals()
```

查看当前帧的全局变量字典

```
p globals()
```

查看对象的所有实例变量（返回 __dict__）

```
p vars(obj)
```

列出对象的所有属性和方法（含继承）

```
p dir(obj)
```

显示表达式类型

```
whatis expression
```

例：<class 'int'> / <class 'list'>

```
whatis x / [1,2]
```

显示源代码（函数/类）。无法获取源码（如内置函数）会报错

```
source expression
```

每次停止时显示表达式（变化时提示）

```
display [expression]
```

停止显示表达式

```
undisplay [expression]
```

显示当前函数返回值。需先执行 r(eturn) 到函数结束，否则报错

```
retval
```

其他功能

执行任意 Python 语句

```
! statement
```

显示帮助（help pdb 显示完整文档）

```
h(elp) [command]
```

退出调试器（程序中止）

```
q(uit)
```

注意事项

PDB 支持 Tab 补全 快捷变量 .pdbrc 文件配置别名

检查当前模块

```
p $__name__
```

检查当前文件位置

```
p $__file__
```

查看函数文档

```
p $__doc__
```

查看异常

```
p $__exception__
```

查看堆栈

```
p $__traceback__
```

实用技巧

进入交互式解释器（带当前帧上下文），适合临时多步探索

```
interact
```

小习惯：先用 where/w 看堆栈，再用 u/d 切到目标帧，然后用 args/locals()/p 观察变量

一行打印变量名和值（Python 3.8+）

```
print(f"{x=}")
```

支持表达式的自动打印

```
print(f"{x+y=}")
```

美化打印复杂数据结构（自动换行缩进）

```
from pprint import pprint; pprint(data)
```

pip 包管理

安装最新版本的包

```
pip install package_name
```

批量安装依赖包

```
pip install -r requirements.txt
```

升级包到最新版本 (简写 -U)

```
pip install --upgrade package_name
```

卸载包 (-y 跳过确认)

```
pip uninstall package_name
```

导出当前环境所有包版本

```
pip freeze > requirements.txt
```

查看包的详细信息，包括安装位置、版本、依赖等

```
pip show package_name
```

列出所有当前环境中的包及其版本

```
pip list
```

设置全局镜像源 (加速下载)

```
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

Conda 环境管理

创建指定Python版本的虚拟环境

```
conda create -n env_name python=3.10
```

激活虚拟环境

```
conda activate env_name
```

列出所有虚拟环境

```
conda env list
```

在当前环境安装包 (-c conda-forge 指定源)

```
conda install package_name
```

删除整个虚拟环境

```
conda remove -n env_name --all
```

列出所有环境中的包及其版本

```
conda list
```

导出跨平台环境配置文件

```
conda env export > environment.yml
```

根据配置文件重建环境

```
conda env create -f environment.yml
```

清理未使用的包和缓存 (释放空间)

```
conda clean -a
```

uv 极速管理

初始化新项目并指定 Python 版本 (类似 npm init)

```
uv init project_name --python 3.11
```

添加依赖并更新锁文件 (自动管理 venv)

```
uv add package_name
```

运行脚本或命令 (自动处理依赖环境)

```
uv run script.py
```

同步项目依赖到环境 (确保一致性)

```
uv sync
```

安装全局命令行工具 (类似 pipx)

```
uv tool install ruff
```

临时运行工具 (无需安装，别名 uv tool run)

```
uvx pycowsay "hello"
```

安装 Python 解释器版本

```
uv python install 3.12
```

调整（锁定）当前项目使用的 Python 版本

```
uv python pin 3.11
```

兼容 pip 的安装命令 (极速模式)

```
uv pip install -r requirements.txt
```