📓 Python Shell – Complete Notes

🐍 What is Python Shell?

The **Python Shell** is an **interactive interpreter**. You type code and see the result **immediately**!

```
$ python
>>>
```

✅ **Good for**: Quick testing, debugging, learning.

⚠️ Not suitable for building full apps or scripts!

🚀 How to Open Python Shell?

💻 Terminal or Command Prompt:

```
python
```

If Python 2.x installed:

```
python3
```

You'll see:

```
Python 3.x.x (default, ...)
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

That's your **Python Shell Prompt!**

🔁 Loops in Python Shell

1️⃣ `for` Loop:

```
>>> for i in range(3):
...     print(i)
```

Output:

```
0
1
2
```

2️⃣ `while` Loop:

```
>>> i = 0
>>> while i < 3:
...     print(i)
...     i += 1
```

🚫 IndentationError in Shell

Python is whitespace sensitive ⚠️

❌ Bad Example:

```
>>> for i in range(3):
>>> print(i)
```

Output:

```
IndentationError: expected an indented block
```

☑️ **Fix**:

```
>>> for i in range(3):
...     print(i)
```

Use **4 spaces or a tab** after `:` colon 📝

☑️ Use Cases of Python Shell

🔧 **Task**	💡 **Use in Shell**
🧪 Quick math	`>>> 2 * (3 + 4)`
🧠 Test logic	`>>> "yes" if True else "no"`

🔧 **Task**	💡 **Use in Shell**
🔍 Debug a function	Paste and run inside shell
📦 Test packages	`import numpy as np`
🛠️ Explore modules	`>>> dir(math)`

🔥 Tips for Using Python Shell

🧩 Use Built-in Help:

```
>>> help(str)
>>> help("modules")
```

🧪 Multi-line editing:

Use `...` to continue lines:

```
>>> def greet():
...     print("Hello")
...     print("World")
```

🕵️ Check memory / variables:

```
>>> a = 10
>>> globals()
>>> locals()
```

🖌️ Clear screen:

In UNIX:

```
Ctrl + L
```

In Windows:

```
cls
```

But inside shell, just restart for a clean screen.

⚙️ Shell in Production? 🏭

While Python shell is **not** directly used in production apps, here are some advanced **real-world scenarios**:

🐳 Docker containers:

Use `python` shell to test scripts in a running container.

🛠️ Debugging live environments:

```
python manage.py shell  # (Django)
```

🔒 Security testing:

Testing scripts in isolated shells (e.g., inside virtualenvs)

📦 Jupyter / IPython:

Shell experience with enhanced features (autocomplete, plots).

🧠 Pro Tips for Shell Usage

✅ Use **`IPython`** for advanced shell (color syntax, magic commands):

```
pip install ipython
ipython
```

✅ Use `venv` to isolate packages:

```
python -m venv env
source env/bin/activate  # or .\env\Scripts\activate on Windows
```

✅ Use `.pythonrc.py` to customize your shell startup behavior (e.g., auto-import numpy).

✅ Combine with `dir()`, `type()`, `id()`, and `print()` to explore objects deeply.

📃 Summary

✅ **Feature**	💬 **Description**
🌀 Python Shell	Interactive prompt to run Python statements
🔁 Loops Supported	`for`, `while` – great for small tests
🚫 IndentationError	Always indent blocks after `:`

☑ **Feature**	💬 **Description**
📦 Use Cases	Testing logic, trying modules, math, learning
💼 Production Tips	Use in testing/debugging within dev workflow

---

🐍 Python Shell – Full Notes with Import, Reload & All 🧠 🚀

🎯 What is Python Shell?

It's an **interactive prompt** where you write Python code and get results instantly! Ideal for **experimentation**, **debugging**, and **learning**.

```
$ python
>>>  # You're now in the Python shell!
```

💾 How to Import Your Python Files in Shell

Suppose you have a Python file like this:

📁 `math_utils.py`

```python
# math_utils.py
PI = 3.14159

def square(x):
    return x * x

def greet(name):
    return f"Hello, {name}!"
```

☑ Import the File:

```
>>> import math_utils
```

🧠 Access Functions and Variables:

```
>>> math_utils.square(5)
25
```

```
>>> math_utils.PI
3.14159

>>> math_utils.greet("Darshan")
'Hello, Darshan!'
```

🪄 Tip: Use `from ... import ...`

```
>>> from math_utils import square, PI
>>> square(4)
16
>>> PI
3.14159
```

⛔ Can't access `greet` now unless imported explicitly.

🗂️ How Python Finds Your File?

Python looks for your file in these paths:

```
>>> import sys
>>> sys.path
```

💡 If your file isn't in current directory:

```
>>> import sys
>>> sys.path.append("/path/to/your/file")
```

Now `import myfile` will work from that location.

🔄 How to Reload Your Imported File on the Go

Suppose you modify your file `math_utils.py` after importing...

Python **doesn't reload it automatically** 😨

✅ Solution:

```
>>> import importlib
>>> importlib.reload(math_utils)
```

Now changes will reflect without restarting the shell! 💥

👀 What Happens Behind the Scenes?

Python caches imported modules:

- First import: Loads and stores in memory
- Re-import: Uses the cached version
- Use `importlib.reload()` to fetch updated version.

✅ Real-Life Shell Use Cases (with Imports)

🧪 **Task**	💬 **Shell Commands**
Test custom function	`import myutils; myutils.say_hi()`
Change file & update in shell	`importlib.reload(myutils)`
Explore new variables	`dir(myutils)` or `vars(myutils)`
Debug logic	`print()` and `type()` inside shell
Validate data before scripting	Write/Run code blocks in shell

---

💼 Production Tips with Shell Imports

✅ Use `manage.py shell` in Django or `flask shell` for live access ✅ IPython shell enhances imports, reloads, autocompletion ✅ Test modules independently before using in actual app ✅ Don't forget to `reload()` after file changes ✅ For testing complex scripts, break into reusable modules

---

🚀 BONUS: Importing with Aliases

```
>>> import math_utils as mu
>>> mu.square(3)
9
```

Clean and readable in bigger scripts!

🧾 Summary Table

🧩 **Feature**	💬 **Explanation**
`import file`	Loads entire module, access with `file.name()`
`from file import`	Load specific members directly

🧩 **Feature**	💬 **Explanation**
`reload(module)`	Updates shell to reflect file changes without restart
`dir()` / `vars()`	Explore what's inside your module
Aliases	`import myfile as mf` for shorter access

🧪 Final Example Session

🗀 `greetings.py`

```
# greetings.py
name = "Darshan"

def hello():
    return f"Hi, {name}"
```

📦 Shell session:

```
>>> import greetings
>>> greetings.hello()
'Hi, Darshan'

# You change name to "Python Master" in file...
# Back to shell:
>>> importlib.reload(greetings)
>>> greetings.hello()
'Hi, Python Master'
```
