COMP1531

🔨 Python
2.3 - Importing & Paths

In this lecture

Why?

- Most python projects involve working across multiple files that need to interact

What?

- How to import
- Different ways to import
- Importing hierarchy

importing and modules

In python, just like C, you're able to import both code from libraries and code from files you write.

```
1 #include <library.h>
2
3 #include "yours.h"
```

Python is similar, though there is less visual distinction

importing and modules

calmath.py

```
1  def daysIntoYear(month, day):
2      total = day
3      if month > 0:
4          total += 31
5      if month > 1:
6          total += 28
7      if month > 2:
8          total += 31
9      if month > 3:
10         total += 30
11     if month > 4:
12         total += 31
13     if month > 5:
14         total += 30
15     if month > 6:
16         total += 31
17     if month > 7:
18         total += 30
19     if month > 8:
20         total += 31
21     if month > 9:
22         total += 30
23     if month > 10:
24         total += 31
25     return total
26
27 def quickTest():
28     print(f"month 0, day 0 = {daysIntoYear(0,0)}")
29     print(f"month 11, day 31 = {daysIntoYear(11,31)}")
30
31 #if __name__ == '__main__':
32 #    quickTest()
33
34 quickTest()
```

importto.py

```
1  import sys
2
3  import calmath
4
5  if len(sys.argv) != 3:
6      print("Usage: importto.py month dayofmonth")
7  else:
8      print(calmath.daysIntoYear(int(sys.argv[1]), \
9                                 int(sys.argv[2])))
```

What is this for??

__name__ is a variable that:

- is "__main__" if the file it's used in is the file being directly invoked by the python interpreter
- Is the name of the module (e.g. "calmath") if the file it's used in is being used via an import from another file

Ways to import

use.py

```python
# Method 1
import * from lib

# Method 2
from lib import one, two, three

# Method 3
import lib
```

lib.py

```python
def one():
    return 1

def two():
    return 2

def three():
    return 3
```

Which ways do we prefer and why?

- Method 1 pollutes the namespace
- Method 2 generally clearest
- Method 3 useful if imported items need context

Relative Imports

```
1 import sys
2 print(sys.path)
3 # or python3 -c "import sys; print(sys.path)"
```

When you import in python, python will look for that module:

1. As one of the built-in binaries
2. Then, relative to the directory your python process was invoked in
3. Then, relative to directories in the sys.path list

For (2), without thinking more deeply about it, this is why it's important in a project you try and invoke files from a consistent directory.

Feedback