Lecture 04

Wednesday, August 21, 2024 2:37 AM

Libraries

Code written by somebody that is useful to you. A lot of code can be modular.

Python Libraries! "random"

Some problems have already been solved, use solutions!

import Keyword

```
import random
random. choice (["heads", "tails"])
print (coin)
```

← .choice() takes a list, chooses one randomly.

Alternate implementation, maybe we only need a specific function only

```
from random import choice
coin = choice (["heads", "tails"])
print (coin)
```

randint()

```
import random
number = random.randomint(1,10)
```

range, inclusive.

shuffle

```
cards = ["J", "Q", "K"]
random.shuffle(cards)
for card in cards:
....print(card)
```

.shuffle() shuffles in place, doesn't return anything.

Statistics Library

```
import statistics
statistics. mean([100, 90])
```

Command Line Arguments

pass arguments to the actual program. sys module has argv, argument vector.

```
import sys

print("hello, my name is", sys.argv[1])
```

program expects user to already give some info.

note, we use python name.py Leo to properly run this program

This program expects an argument, for comepleteness we should handle the exception

```
import sys

try:
.... print("...", sys.argv[1])
except IndexError:
.... print(" Too few arguments
```

Or, more defensively, do:

```
if len(sys.argv)<2:
.... print("two few args")
elif len(sys.argv)>2:
.... print(" too many args")
else:
.... print(" hello, my name is argv[1]
```

argv[0] is the title of the program.

This is neat, but a little excessive. Exit prematurely on errors.

```
import sys
if len(sys.argv)<2:
....sys.exit("two few args")
elif len(sys.argv)>2:
....sys.exit(" too many args")

print(" hello, my name is ar v[1]")
```

early exit!

Let's adjust for many given inputs

```
import sys

if len(sys.argv) < 2
.... sys. exit(" to few args")

for arg in argv [1:]
....print("hello, ", arg)
```

start @ 1, not Ø

We're able to take many inputs

Packages

Third party libraries, one is cowsay. install it with pip

```
pip install cowsay
```

Now we can use it

```
import cowsay
import sys

if len(sys.argv) == 2:
.... cowsay.cow("hello, ", sys.argv[1])
```

could also use .trex.

Mostly just a demonstration of the power and ease of use to use other's code.

Why use commandline arguments? can speed up certain workflows,

APIs

We can connect to 3rd party APIs that give use useful data by "pretending" to be a browser.

The library helps us accomplish this

JSON is the format in which the data we fetch is often returned in.

```
import requests
import sys

if len(sys != 2):
.... sys.exit()

response = requests.get(".....api_endpoint.."+ sys.argv[1]
print(response.json())
```

Python converts received json data to dict

```
import json
print(json.dumps(response.json(), indent=2))
```

helps formatting further

A dict can be within a dict.

```
o = response.json()
for result in o["results"]:
....print(result["trackName"]:
```

We know to use these keys because we understan the JSON response

Make Your Own...

```
"sayings.py"
def hello(name):
....

def goodbye(name):
....

def main:
....
```

```
"say.py"
from sayings import hello
```

Long story short, don't unconditionally call main() in a file you intend on importing elsewhere.

```
if __name__ == "__main__":
.... main()
```

special keyword that allows us to restrict when main() in "sayings.py" is called.