“Me just met you and this is crazy,
but you got cookie, so share it maybe?”
- C. Monster

N: Flask's **request object** stores info about incoming request

...its useful fields:

`request.headers` HTML headers from client browser

`request.method` GET or POST

`request.args`

* arguments, as a querystring from GET request
* immutable dictionary

`request.form`

* arguments sent via POST request
* immutable dictionary

if this is in your HTML file:

```
...
<body>
  <form action="/auth">
    <input type="text" name="username">
    <input type="submit" name="sub1">
  </form>
</body>
...
```

this py code will handle responding to it:

```
@app.route("/auth")
def authenticate():
    return "Waaaa hooo HAAAH"
```

PROTIP: **diagnostic print statements** (powerful magicks!)

eg,

```
@app.route("/auth")
def authenticate():
    print(app)
    print(request)
    print(request.args)
    return "Waaaa hooo HAAAH"
```

PROTIP: **diagnostic print statements** (powerful magicks!)

eg,

```
@app.route("/auth")
def authenticate():
    print(“////////////!!!\\\\\\\\\\\\\\\”)
    print(app)
    print(“!!!!!!!!!!!!!!!!”)
    print(request)
    print(“================================”)
    print(request.args)
    return "Waaaa hooo HAAAH"
```

COOKIE:

* small file given by a website
  to your web browser
    for storage on your LOCAL machine

* useful for maintaining awareness of identity
  across multiple page visits on same site

* transmitted with request

Q:

How does one GET COOKIES?

A: Ask flask for them!

`request.cookies.get( KEY )`

* accesses cookie data if available
* (no error thrown if key missing)

eg,

`request.cookies.get('username')`

is equivalent to

`request.form['username']`

but will not throw Key error if username not found...

C is for Cookie, but that's not good enough for me...

S is for **SESSION**:

Flask module to facilitate “remembering” users
from one request to the next.

S is for **SESSION**:

* Flask module to facilitate “remembering” a user from one request to the next.
* Flask session data is *maintained* by server but *stored* in cookies on client.
* cookie payload is encoded
* cookie is **securely signed**
* any http(!s) transmission is “sniffable”
* Flask **session object** works like a dict

Flask session data is *maintained* by server
but *stored* in cookies on client.

* cookie payload is encoded
* any HTTP transmission is “sniffable”
  (...so is an HTTPS transmission, but `#;jlkjasdf$3j*74,as`)
* Flask **session object** works like a dict
* cookie is **securely signed**
  (similar to your GH pvt/pub key mechanism)

N: on the security of **Flask session cookies**...

Data is *encoded* – not *encrypted*.

an encoding scheme (eg ASCII, Unicode) ensures
 data xfr, consumption w/ reliability/repeatability/integrity
 * 2-way, anyone who knows schema can go either way

an encryption algo (eg AES, RSA) ensures data privacy
 * need private key to unlock transmission

A stranger can (with a bit of work) view this cookie
 data – **but not change it**
 (...unless she knows secret key used to sign it)

Q: What step must you, bright Devo,
do to ensure a private session?

A: Generate a private key, and assign as
built-in secret_key

app.secret_key = <randomly_generated_string>

to get a randomized string:

```
import os
os.urandom(32)
```
-> 32 bits of random data as a string

MAXIM:

Never, never, never,
NEVER, NEVER, NEVER
store a private key in a publicly-viewable location

(*...like your workshop*)

Eg,

```
from flask import session
```

...

to add data to a session:

```
session[KEY] = value
```

to remove data from a session:

```
session.pop(key)
```

N re: py modules:

If **`ants.py`** exists in current working dir,

```
import ants
```

...will make functions defined in `ants.py` available for use like this:

```
ants.foo()
```

BUT your support fxns will live in `/utl`

…so use Python **package** functionality...

This will make python view `utl` *as a package*:

```
path/to/flask_app$
 touch utl/__init__.py
```

N:

* **`__init__.py`** is a special file that
  flags a dir as a *package* dir
* **`__init__.py`** can (and should) be empty for now

N: with these files in `/utl` …

`__init__.py`

`ants.py`

This code in app.py will load module *ants* from package *utl*:

```
import utl.ants as ants
```

or

```
from utl import ants
```

...and thus make functions defined in `ants.py` available for use like so:

```
ants.foo()
```

TASK (at earliest opportunity):

Test drive this functionality.

TASK (HW):

```
}
```

:

$.

:

$.

N re: py modules:

If **`ants.py`** exists in current working dir,

```
import ants
```

...will make functions defined in `ants.py` available for use like this:

```
ants.foo()
```

BUT your support fxns will live in `/utl`

…so use Python **package** functionality...

This will make python view `utl` *as a package*:

```
path/to/flask_app$
 touch utl/__init__.py
```

N:

* **`__init__.py`** is a special file that
  flags a dir as a *package* dir
* **`__init__.py`** can (and should) be empty for now

N: with these files in `/utl` …

`__init__.py`

`ants.py`

This code in app.py will load module *ants* from package *utl*:

```
import utl.ants as ants
```

or

```
from utl import ants
```

...and thus make functions defined in `ants.py` available for use like so:

```
ants.foo()
```

TASK (at earliest opportunity):

Test drive this functionality.

Flask session data is *maintained* by server
but *stored* in cookies on client.

* cookie payload is encoded
* any HTTP transmission is “sniffable”
  (...so is an HTTPS transmission, but `#;jlkjasdf$3j*74,as`)
* Flask **session object** works like a dict
* cookie is **securely signed**
  (similar to your GH pvt/pub key mechanism,
  or a download file checksum)

N: on the security of **Flask session cookies**...

Data is *encoded* – not *encrypted*.

an encoding scheme (eg ASCII, Unicode) ensures
 data xfr, consumption w/ reliability/repeatability/integrity
 * 2-way, anyone who knows schema can go either way

an encryption algo (eg AES, RSA) ensures data privacy
 * need private key to unlock transmission

A stranger can (with a bit of work) view this cookie
 data – **but not change it**
 (...unless she knows secret key used to sign it)

to generate a private key and assign as built-in secret_key

app.secret_key = <randomly_generated_string>

to get a randomized string:

```
import os
os.urandom(32)
```
-> 32 bits of random data as a string