DEFAULT VALUES

What happens at run-time…

When a module is loaded: all code is executed immediately

Module Code

`a = 10`	the integer object `10` is created and `a` references it
`def func(a):` `  print(a)`	the function object is created, and `func` references it
`func(a)`	the function is executed

What about default values?

Module Code

```
def func(a=10):
    print(a)
```

the function object is created, and func references it

the integer object 10 is evaluated/created
and is assigned as the default for a

```
func()
```

the function is executed

by the time this happens, the default value for a has already been evaluated and assigned – it is not re-evaluated when the function is called

So what?

Consider this:

We want to create a function that will write a log entry to the console with a user-specified event date/time. If the user does not supply a date/time, we want to set it to the current date/time.

```
from datetime import datetime

def log(msg, *, dt=datetime.utcnow()):
    print('{0}: {1}'.format(dt, msg)
```

```
log('message 1')    → 2017-08-21 20:54:37.706994 : message 1
```

a few minutes later:

```
log('message 2')    → 2017-08-21 20:54:37.706994 : message 2
```

Solution Pattern

We set a default for `dt` to `None`

Inside the function, we test to see if `dt` is still `None`

if `dt` is `None`, set it to the current date/time

otherwise, use what the caller specified for `dt`

recall that this is equivalent to:

```
if not dt:
    dt = datetime.utcnow()
```

```
from datetime import datetime

def log(msg, *, dt=None):
    dt = dt or datetime.utcnow()
    print('{0}: {1}'.format(dt, msg)
```

In general, always beware of using a mutable object (or a callable) for an argument default

Code