Super Secret Advanced Ninja Python Techniques

Advanced Python tricks

Things you hear about tonight will make your code shorter, will reduce code duplication, and will help with abstractions.

However, they are tools on top of what you already know. You can always do without them and still get the job done.

How would you implement this spec?

```
def remove_evens(lst):
    """ lst is a list of integers.
    Returns a new list with the even
    numbers removed. """
    pass

odds = remove_evens(lst)
```

A loop, a conditional, an append

```
def remove_evens(lst):
    new_lst = []
    for elem in lst:
        if elem % 2 == 1:
            new_lst.append(elem)
    return new_lst

odds = remove_evens(lst)
```

List comprehensions to the rescue!

```
odds = [x for x in lst if x % 2 == 1]
```

...that's it. Just one line.

Comprehending list comprehension

```
odds = [x for x in lst if x % 2 == 1]
```

Three essential parts:

1. The loop
2. The filter
3. The expression

Part 1: The loop

```
odds = [x for x in lst if x % 2 == 1]
```

The highlighted line says to loop over all the elements in lst.

For each element, filter it.

If it passes the filter, add the expression to the final list.

Part 2: The filter

```
odds = [x for x in lst if x % 2 == 1]
```

The filter is highlighted above.

If x is as element in lst, then x % 2 != 1 has to be True in order for it to be added to the final list. If it is not True, then this element is skipped.

(The filter is also optional.)

Part 3: The Expression

```
odds = [x for x in lst if x % 2 == 1]
```

The first "x" in the square brackets is the expression. If a given element x passes the filter, then the expression x gets added to the final list.

In this case, x is just the element itself, so any element in the original lst that passes the filter will get added directly without modification.

What is x?

```
odds = [x for x in lst if x % 2 != 1]
```

x here is just a variable name. It could be "elem" or whatever you'd like. It's like declaring a new variable when you say "for **x** in my_list:"

You use this variable name in the filter and expression.

List Comprehension Recap

```
[x for x in lst if x % 2 != 1]
```

[(expression) (loop) (filter)]

That's not really the whole story

Technically, in order:

- 1 expression
- 1 loop
- any number of loops or filters in any order

For example:

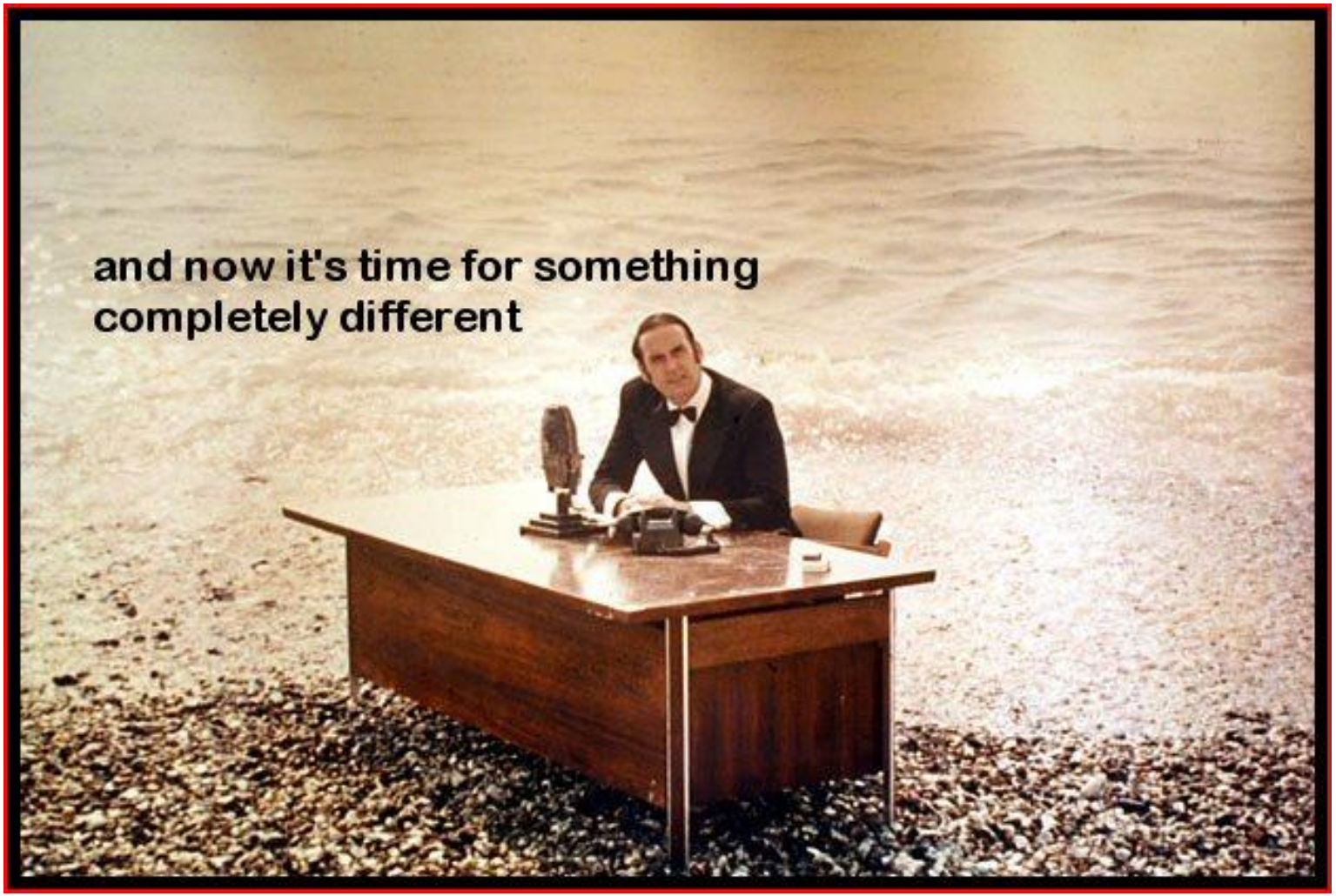
```
[(x, y) for x in range(3) if x != 1 for y in range(3) if y != 1]
```

Expanding a list comprehension

```
p =[(x, y) for x in range(3) if x != 1 for y in range(3) if y != 1]
```

```
p = []
for x in range(3):
    if x != 1:
        for y in range(3):
            if y != 1:
                p.append((x, y))
```

phew



Sooo...what's going on here?

```
def f(x):
    return x * 2

def printer(func, value):
     print func(value)

for x in range(10):
    printer(f, x)
```

What happened?

```
def printer(func, value):
     print func(value)

for x in range(10):
    printer(f, x)
```

The variable func is actually an object, representing the function f! And we pass it to the printer function, which calls f via func, even though it knows nothing about f.

Python interpreter will tell you this

```
>>> print type(f)
<type 'function'>

>>> print f
<function f at 0x7f94e4cca2a8>

>>> f2 = f
>>> print f2
<function f at 0x7f94e4cca2a8>
```

What does def func(): do, really?

```
def my_function(x, y):
    pass
```

The def keyword actually assigns a function object in the local environment to the variable named `my_function`.

Passing a different function

```
def other_function(x):
    return x < 3

def printer(func, value):
     print func(value)

for x in range(10):
    printer(other_function, x)
```

Works like any other kind of object

```
def hello(name):
    print "Hello, %s!" % name
def hola(name):
    print "Hola %s!" % name
def greeting_func():
    return random.choice([hello, hola])

greet = greeting_func()
greet("enne")
```

Built-in functions on functions!

list2 = map(function, list)

map takes a function and a list and returns a new list that's the result of calling that function on each element in that list.

A quick map example

```
def greater_than_three(x):
    return x > 3
result = map(greater_than_three, range(6))
```

What is the variable result after this code?