Lab5_sol

December 7, 2022

1 11 Python Tricks that makes coding easier.

The text cells support Markdown as well as plain text.

1.1 01 Enumerate

If you really want to index a list on which you are executing a for loop, you have enumerate method to make it easier.

```
[ ]: values = ['a', 'b', 'c']

     # typical python for loop
     for v in values:
         print(v)
```

```
a
b
c
```

```
[ ]: # this is not the best way to index the list inside a for loop

     i = 0
     for _ in values:
         print(i, values[i])
         i += 1

     # Nor this
     for i in range(len(values)):
         print(i, values[i])
```

```
0 a
1 b
2 c
0 a
1 b
2 c
```

[ ]:
```python
# instead you can use enumerate()
for i, v in enumerate(values):
    print(i, v)
```

```
0 a
1 b
2 c
```

[ ]:
```python
# Now write a function using enumerate that takes two arrays with the same␣
 ↪length and creates
# a paired array as a result. [1, 2, 3], ['a', 'b', 'c'] -> [[1,'a'], [2,'b'],␣
 ↪[3,'c']]

def zipper(arr1, arr2):
    # send error if the arrays don't have the same length
    assert len(arr1) == len(arr2)

    # create an empty array for the result
    res = []

    for i, a1 in enumerate(arr1):
        res.append([a1, arr2[i]])
    return res

array1 = [1,2,3]
array2 = ['a', 'b', 'c']
print(zipper(array1, array2))
```

```
[[1, 'a'], [2, 'b'], [3, 'c']]
```

1.2 02 Else block

As mentioned in the lecture, we can use `else` blocks along with many flow control keywords in Python.

[ ]:
```python
# this method checks if a number is prime
def prime(num):
    if num > 1:
        for i in range(2,num):
            if (num % i) == 0:
                return False
        else:
            return True

# now write a method that checks if there is any prime numbers in a list
def list_prime(arr):
    for a in arr:
        if prime(a):
            return True
    else:
        return False

print(list_prime([4,6,8]))
```

```
False
```

1.3 03 List comprehensions

[ ]:
```
# we can write more compact code using Python list comprehension
# let's assume that we want to save the result of an elementwise operation on a␣
 ↪list

x = [1, 2, 3, 4, 5]
print(x)

# now we want to store y = x^2 - 3
y = [n**2 - 3 for n in x]
print(y)
```

```
[1, 2, 3, 4, 5]
[-2, 1, 6, 13, 22]
```

[ ]:
```
# now define a method that returns the cartesian product of two lists using␣
 ↪list comprehension
def cart_prod(arr1, arr2):
    return [[a, b] for a in arr1 for b in arr2]

print(cart_prod([1,2,3], ['a', 'b', 'c', 'd']))
```

```
[[1, 'a'], [1, 'b'], [1, 'c'], [1, 'd'], [2, 'a'], [2, 'b'], [2, 'c'], [2, 'd'],
[3, 'a'], [3, 'b'], [3, 'c'], [3, 'd']]
```

1.4 04 Formatted Strings

There are multiple ways to inject your variables inside a python string but here is the most straight forward one. You should put an `f` character before starting your string, and put your variables inside `{}`.

[ ]:
```
fname, lname = 'Mohammad', 'Heydari'
print(f'My first name is:{fname}, and my last name is:{lname}.')
```

```
My first name is:Mohammad, and my last name is:Heydari.
```

[ ]:
```
# now define a function that takes a dictionary schedule and prints it properly.
# here is a sample schedule
schedule = {"laundry": 50, "study": 120, "dinner": 30}
```

```
def print_schedule(dict1):
    for k, v in dict1.items():
        print(f'activity {k}, takes: {v} minutes.')

print_schedule(schedule)
```

```
activity laundry, takes: 50 minutes.
activity study, takes: 120 minutes.
activity dinner, takes: 30 minutes.
```

1.5 05 Itertools

`Itertools` is a library available in any Python installation providing you with functions for efficient looping. documentation

[ ]:
```
import itertools

for a, b in itertools.product(['a', 'b', 'c'], range(1,5)):
    print(f'{{{a}, {b}}}')
```

```
{a, 1}
{a, 2}
{a, 3}
{a, 4}
{b, 1}
{b, 2}
{b, 3}
{b, 4}
{c, 1}
{c, 2}
{c, 3}
{c, 4}
```

[ ]:
```
import itertools

# now write a method to extract the all of the possible DNA strings with length␣
 ↪n
# product method has a 'repeat' argument that controls the length of the output
# dna chains are shown using A,T,C,or G characters.
# for instance AACTCGAG is a dna string with length 8.

def dna_counter(n):
    for dna in itertools.product('ATCG', repeat=n):
        print(dna)

dna_counter(2)
```

```
('A', 'A')
('A', 'T')
('A', 'C')
('A', 'G')
('T', 'A')
('T', 'T')
('T', 'C')
('T', 'G')
('C', 'A')
('C', 'T')
('C', 'C')
('C', 'G')
('G', 'A')
('G', 'T')
('G', 'C')
('G', 'G')
```

1.6 06 One-liner if else

You can use mini conditional statement to initiate your variables or execute other functions.

[ ]:
```
age = 45
if age>40:
    print('old')
else:
    print('young')
```

```
old
```

[ ]:
```
age = 45
print('old' if age>40 else 'young')
```

```
old
```

[ ]:
```
import random
# random.random() gives you a random value between [0, 1).
# use this to initiate your variable x with 0 if the random value is smaller␣
 ↪than 0.2
# or 1 otherwise.
x = 0 if random.random()<0.2 else 1
print(x)
```

```
1
```

1.7 07 Pathlib

Pathlibs enables us to find the current directory when executing our code. It does not depend on the OS or the libraries you have installed on your python distribution.

[ ]:
```python
import pathlib
print(f'currnet working directory: {pathlib.Path.cwd()}')
```

```
currnet working directory: /content
```

[ ]:
```python
# using pathlib you can find out if a path is pointing to a director or a file.
# write a code using pathlib.Path.is_dir to verify that
# your current working directory is actually a folder

pathlib.Path.is_dir(pathlib.Path.cwd())
```

[ ]:
```
True
```

1.8 08 Decorators (Wrapper Functions)

It is one of the most powerful tools of Python programming specifically in Django. Most of the time we just use predefined decorators but it is beneficial to know how to implement one. Decorators are functions that take another function as an input, consider it as a black box and add more functionality to it.

[ ]:
```python
# we want to write a decorator that makes its following function to run twice
def do_twice(func):
    def wrapper_func():
        func()
        func()
    return wrapper_func

# now this is the way to use a decorator:
@do_twice
def pr_name():
    print('Hey, Mohammad!')

# interpret the code above as this:
# pr_name = do_twice(pr_name)
# now we only have to call the function being wrapped once:
pr_name()
```

```
Hey, Mohammad!
Hey, Mohammad!
```

[ ]:
```python
# now let's write a decorator to report excution time
import time

# time.time() returns the current system time and you have to substract its
# value in the beginning of a process from its value in the end of a process

def time_takes(func):
```

```
# *args and **kwargs passes every arguments from the original
# function to the wrapped one
    def wrap(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()

        print(f'{func.__name__}, takes {end-start} seconds.')
        return result
    return wrap

@time_takes
def lazy():
    for _ in range(3):
        time.sleep(1)

lazy()
```

```
lazy, takes 3.0032331943511963 seconds.
```

The last three tricks help us work with lists without iterating over them.

1.9 09 Map

map takes a function and a list as its arguments. Then executes the function on each list element.

[ ]:
```
# let's write a map to extract the length of each elements in a list of strings
array = ['hello', 'world', 'bit2008']
lengths = map(len, array)
print(list(lengths))
```

```
[5, 5, 7]
```

[ ]:
```
# let's write another map to calculate the elementwise sum of two arrays
arr1 = [1, 3, 12, -9, 4, -5, 8]
arr2 = [3, -4, 5, -6, 7, -9, 8]

def sum(a, b):
    return a+b

y = map(sum, arr1, arr2)
print(list(y))
```

```
[4, -1, 17, -15, 11, -14, 16]
```

1.10 10 Filter

similar to `map`, filter takes a function to pass some of the elements of its other argument which is a list and remove the rest.

```
[ ]: # lets write a filter map that removes frequencies below 100 Hz
     freqs = [10, 120, 125, 230, 80, 150, 70]
     passed = filter(lambda x: True if x>100 else False, freqs)
     print(list(passed))
```

```
[120, 125, 230, 150]
```

1.11 11 Zip

Zip matches the corresponding elements of multiple lists together.

```
[ ]: id = (1,2,3,4,5,6)
     name = ('Alex', 'Joe', 'Mika')
     major = ('Music', 'Comp Sci', 'IT')
     res = zip(id, name, major)
     print(list(res))
```

```
[(1, 'Alex', 'Music'), (2, 'Joe', 'Comp Sci'), (3, 'Mika', 'IT')]
```

1.12 References

An overview of 10+1 Python features that only advanced programmers use