01ex_Fundamentals

March 14, 2020

1. Write the following as a list comprehension

```
In [1]: # 1
        ans = []
        for i in range(3):
            for j in range(4):
                ans.append((i, j))
        print (ans)

        ansc=[(x,y) for x in range(3) for y in range(4)]
        print (ansc)

        # 2
        ans = map(lambda x: x*x, filter(lambda x: x%2 == 0, range(5)))
        print (list(ans))

        ansc = [x*x for x in range (5) if x%2 == 0]
        print (ansc)
```

```
[(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (1, 3), (2, 0), (2, 1), (2, 2), (2, 3
[(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (1, 3), (2, 0), (2, 1), (2, 2), (2, 3
[0, 4, 16]
[0, 4, 16]
```

2. Convert the following function into a pure function with no global variables or side effects

```
In [2]: x = 9
        def f(alist):
            for i in range(x):
                 alist.append(i)
            return alist

        def f2(alist):
            alist=[]
            for i in range(x):
                alist.append(i)
            return alist
```

```
        alist = [1,2,3]
        ans = f2(alist)
        print (ans)
        print (alist) # alist has not been changed!
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3]
```

3. Write a `decorator` hello that makes every wrapped function print "Hello!", i.e. something like:

```
@hello
def square(x):
    return x*x
```

```
In [3]: def hello_decorator(f):
            def wrapper(*args, **kw):
                print("hello!")
                f(args[0])
                print("decoratore carino")
            return wrapper
        def hello_decorator_noarg(f):
            def wrapper():
                print("hello!")
                f()
                print("decoratore carino")
            return wrapper
        @hello_decorator
        def f1(x):
            print (x*x)
        @hello_decorator_noarg
        def f2():
            print("miao")

        f1(5)
        f2()
```

```
hello!
25
decoratore carino
hello!
miao
decoratore carino
```

4. Write the factorial function so that it a) does and b) does not use recursion.

```
In [4]: def fact_iter(x):
            t=1
```

```
        for i in range (1,x+1):
            t*=i
        return t

    def fact_rec(x):
        if (x==0 or x==1):
            return x
        else:
            return x*fact_rec(x-1)

    x=5
    print(fact_iter(5))
    print(fact_rec(5))
```

```
120
120
```

5. Use HOFs (zip in particular) to compute the weight of a circle, a disk and a sphere, assuming different radii and different densities:

```
densities = {"Al":[0.5,1,2],"Fe":[3,4,5],"Pb": [15,20,30]}
radii = [1,2,3]
```

where the entries of the dictionary's values are the linear, superficial and volumetric densities of the materials respectively.

In particular define a list of three lambda functions using a comprehension that computes the circumference, the area and the volume for a given radius.

```
In [5]: import math
        densities = {"Al":[0.5,1,2],"Fe":[3,4,5],"Pb": [15,20,30]}
        radii = [1,2,3]

        def ring(r, d):
            l=2.0*math.pi*r
            w=l*d
            return w

        def disc(r, d):
            l=math.pi*r**2
            w=l*d
            return w

        def sphere(r, d):
            l=(4.0*math.pi*r**3)/3.0
            w=l*d
            return w

        form=[ring, disc, sphere]
```

```
        shape=["Circle : ","Disk   : ","Sphere : "]

        for name,dens in zip(densities.keys(),densities.values()):
            print (name, " masses:")
            for rad in  range(len(radii)):
                print ("Radius: ", rad+1)
                for d0,d1,density in  zip(shape,radii,dens):
                        mass = form[d1-1](rad+1,density)
                        print (d0, round(mass,2))
```

```
Al  masses:
Radius:  1
Circle :  3.14
Disk   :  3.14
Sphere :  8.38
Radius:  2
Circle :  6.28
Disk   :  12.57
Sphere :  67.02
Radius:  3
Circle :  9.42
Disk   :  28.27
Sphere :  226.19
Fe  masses:
Radius:  1
Circle :  18.85
Disk   :  12.57
Sphere :  20.94
Radius:  2
Circle :  37.7
Disk   :  50.27
Sphere :  167.55
Radius:  3
Circle :  56.55
Disk   :  113.1
Sphere :  565.49
Pb  masses:
Radius:  1
Circle :  94.25
Disk   :  62.83
Sphere :  125.66
Radius:  2
Circle :  188.5
Disk   :  251.33
Sphere :  1005.31
Radius:  3
Circle :  282.74
Disk   :  565.49
```

```
Sphere :  3392.92
```

6. Edit the class defintion to add an instance attribute of is_hungry = True to the Dog class. Then add a method called eat() which changes the value of is_hungry to False when called. Figure out the best way to feed each dog and then output "My dogs are hungry." if all are hungry or "My dogs are not hungry." if all are not hungry. The final output should look like this:

```
   I have 3 dogs.  Tom is 6.  Fletcher is 7.  Larry is 9.  And they're all
mammals, of course.  My dogs are not hungry.
```

```
# Parent class
class Dog:

    # Class attribute
    species = 'mammal'

    # Initializer / Instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def description(self):
        return "{} is {} years old".format(self.name, self.age)

    # instance method
    def speak(self, sound):
        return "{} says {}".format(self.name, sound)

# Child class (inherits from Dog class)
class RussellTerrier(Dog):
    def run(self, speed):
        return "{} runs {}".format(self.name, speed)

# Child class (inherits from Dog class)
class Bulldog(Dog):
    def run(self, speed):
        return "{} runs {}".format(self.name, speed)
```

```
In [6]: class Dog:

            # Class attribute
            species = 'mammal'

            # Initializer / Instance attributes
            def __init__(self, name, age):
                self.name = name
                self.age = age
```

```
            self.is_hungry=True

        # instance method
        def description(self):
            return "{} is {} years old".format(self.name, self.age)

        # instance method
        def speak(self, sound):
            return "{} says {}".format(self.name, sound)
        def status(self):
            if self.is_hungry:
                return "{} is hungry".format(self.name)
            else:
                return "{} is not hungry".format(self.name)
        def eat(self):
            self.is_hungry=False

    # Child class (inherits from Dog class)
    class RussellTerrier(Dog):
        def run(self, speed):
            return "{} runs {}".format(self.name, speed)

    # Child class (inherits from Dog class)
    class Bulldog(Dog):
        def run(self, speed):
            return "{} runs {}".format(self.name, speed)

    def hunger (dogs):
        t,f =0,0
        for d in dogs:
            if d.is_hungry==True:
                t+=1
            else:
                f+=1
        if f==0:
            return "All my dogs are hungry"
        elif t==0:
            return "All my dogs are not hungry"
        else :
            return "My dogs are in a quantum sovrapposition"
```

In [7]:
```
import random
x=random.randrange(4,10)


doggos=[Dog("doggo{}".format(y),random.randrange(5)) for y in range (1,x+1)]
```

```
        print (hunger(doggos))
        for doggo in doggos:
            print(doggo.description())
            #print(doggo.speak("bark"))
            print(doggo.status())
            doggo.eat()
        print (hunger(doggos))
        doggos[0].is_hungry=True
        print (hunger(doggos))

        #for doggo in doggos:
         #   print(doggo.description())
           # print(doggo.speak("bark"))
           # print(doggo.status())
          #  doggo.eat()
```

```
All my dogs are hungry
doggo1 is 4 years old
doggo1 is hungry
doggo2 is 3 years old
doggo2 is hungry
doggo3 is 1 years old
doggo3 is hungry
doggo4 is 2 years old
doggo4 is hungry
doggo5 is 4 years old
doggo5 is hungry
doggo6 is 1 years old
doggo6 is hungry
doggo7 is 4 years old
doggo7 is hungry
doggo8 is 2 years old
doggo8 is hungry
All my dogs are not hungry
My dogs are in a quantum sovrapposition
```

In [ ]:

In [ ]: