Bilişimsel biyolojide nesne tabanlı programlama

Prosedürler ile programlama:

```
a = 8 + 11
print(a)
b = 8.0
b = 'atgtcgtagctggcgtagcggcc'
print(len(b))
```

```
19
23
```

```
# kulağımızı tersten göstermek istersek,
# tam da böyle bir for loop kurarız:
for i in range(0, len(b)):
    print(b[i])
```

```
a
t
g
t
c
g
t
a
g
c
t
g
g
c
g
t
a
g
c
g
g
c
c
```

```
# kulağımızı tersten göstermek istersek,
# tam da böyle bir for loop kurarız:
for i in range(0, len(b)):
    print(b[i])
```

```
a
t
g
t
c
g
t
a
g
c
t
g
g
c
g
t
a
g
c
g
g
c
c
```

VS

```
# pythonic way of looping...
a = t = c = g = 0
for i in b:
    if i == 'a':
        a = a + 1
    if i == 't':
        t = t + 1
    if i == 'c':
        c = c + 1
    if i == 'g':
        g = g + 1
print(a + t)
print(g + c)
print('a+t: %d\tg+c: %d'%(a+t, g+c))
```

```
8
15
a+t: 8  g+c: 15
```

Her seferinde baştan yazmak yerine:

```
# code reuse... algoritmamızı bir fonksiyon haline getirirsek
# her AT-GC hesaplamak istediğimizde bu fonksiyonu çağırabiliriz:
def calcAtgc(sequence):
    a = t = c = g = 0
    for i in sequence:
        if i == 'a':
            a = a + 1
        if i == 't':
            t = t + 1
        if i == 'c':
            c = c + 1
        if i == 'g':
            g = g + 1
    return a+t, g+c # değerleri sırası ile döndür
    # birden fazla değer döndürebilmek çok ultrasonik bir özellik!!!
```

Kodu yeniden kullanmak için fonksiyon tanımlasak

```python
# code reuse... algoritmamızı bir fonksiyon haline getirirsek
# her AT-GC hesaplamak istediğimizde bu fonksiyonu çağırabiliriz:
def calcAtgc(sequence):
    a = t = c = g = 0
    for i in sequence:
        if i == 'a':
            a = a + 1
        if i == 't':
            t = t + 1
        if i == 'c':
            c = c + 1
        if i == 'g':
            g = g + 1
    return a+t, g+c # değerleri sırası ile döndür
    # birden fazla değer döndürebilmek çok ultrasonik bir özellik!!!
```

```python
# fonksiyonu çağıralım:
at, gc = calcAtgc('atgcatgc') # fonksiyonun geri döndürdüğü değerler
# sırası ile değişkenlere atanır
print(at, gc)
```

```
4 4
```

```python
# calcAtgc() fonksiyonunu Tm hesaplayan başka bir
# fonksiyonun içinden çağırmak çok akıllıca
# değil mi? :)))
def calcTm(sequence):
    at, gc = calcAtgc(sequence)
    tm = 4 * gc + 2 * at
    return tm
```

```python
# Tm hesaplamak için:
print(calcTm(b))
```

```
76
```

Daha çok… daha çok...

```
# ya da bir dizi oligomuz olsa...
oligos = ['atgcatgc',
          'ttttttttttt',
          'atatatatatatatatatatat',
          'gcgatacgcatagcgata']
for o in oligos:
    print(calcTm(o))
# listedeki her olgo için döngü fır fır fır...
```

```
24
24
44
54
```

Daha etkin “kod yeniden kullanımı”

Sınıf - Class

Nesne - Instance

Metod - Method

Özellik - Property

Yapıcı - Constructor

Daha etkin “kod yeniden kullanımı”

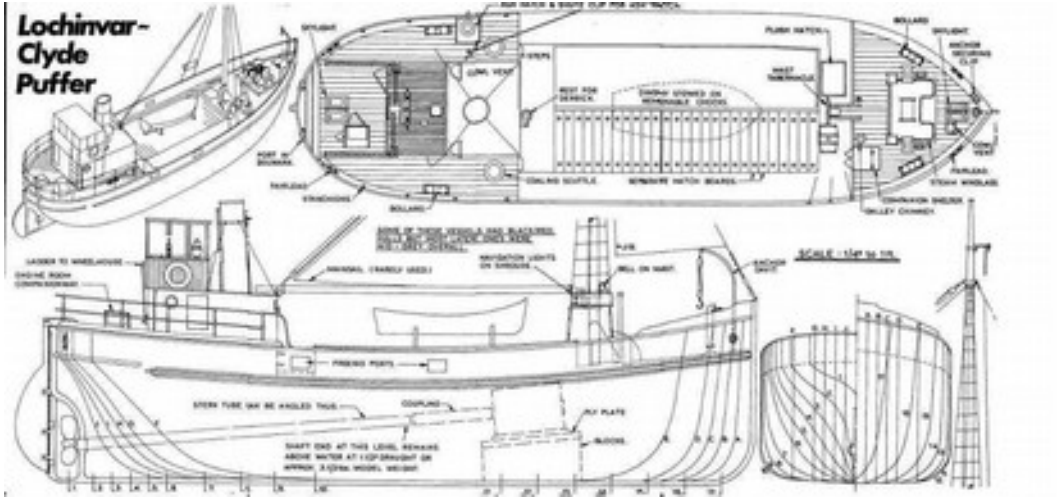
```
# o halde nesne tabanlı yazılım geliştirmeye geçebiliriz...
class pipette(object):

    def __init__(self):
        print('pipette created...')
```

```
p1 = pipette()
```

```
pipette created...
```

Class – kağıttaki plan, yüzmez!!!



Constructor

Object – yüzer :)