INF 443
Dağıtık Sistemler ve Uygulamaları
Laboratuvar Kitapçığı

Serhan Daniş

24 Kasım 2020

İçindekiler

Bölüm 0

Laboratuvara giriş - 14.10.2020

0.1 Kitapçık hakkında

Bu kitapçık dönem ilerledikçe enecektir. Güncel hali aralıklarla öğrencilere iletilecektir.

Bu kitapçık sadece laboratuvar çalışması hakkındadır.

0.2 Platform

python dilinin taşınabilirlik özelliği düşünüldüğünde belirli bir platform kısıtlaması olmaması gerekiyor. Dolayısıyla öğrenciler rahat oldukları işletim sistemlerini kullanabilirler.

Ancak laboratuvar çalışmasında kişisel bilgisayarlarda kurulu bir python alt yapımızın olması gerekiyor.

0.3 Ödev kuralları

1. Ödevler mümkün olduğunca ders saatlerinde yapılmak üzere tasarlanacaktır.
2. Ödevler öğrencinin Github.com üzerindeki kişisel (dışarıya kapalı) hesabından çekilip okunacaktır.
3. Ödevlerin son teslim tarihleri bulunmaktadır.

4. Ödevler kişiseldir. Çalıntı veya kopya yakalanması durumunda **eksi not** kuralı uygulanacaktır.

Bölüm 1

python'a Giriş I - 14.10.2020

Kaynak: An Informal Introduction to python

1.1 Tanıtım

1.1.1 python nedir ve neden kullanıyoruz?

- Yorumlanan (ve hazır derlenmiş), nesne temelli, yüksek seviye bir programlama dili. (Compiled vs. Interpreted)
- python kıvrak ve dinamik bir söz dizimi vardır.
- Paragraflama çok önemlidir ve etki alanını (scope) belirler. C'deki çengelli parantez {} işlevini böyle sağlıyoruz.
- İşletim sistemlerinin önemli parçalarında kullanılıyor.
- Söz dizimi sade, okunaklı ve temizdir.
- Kolay öğrenilebilir, dolayısıyla program geliştirme süreci hızlandırılabilir.
- Farklı platformlarda değişiklik yapmadan veya çok az bir değişiklikle kullanılabilir. (cross-platform portability)
- Ayrı bir derleyiciye ihtiyaç duymaz.

1.1.2 Trivia

- Baş geliştirisi Hollanda'lı Guido Van Rossum.
- İsmi direkt olarak piton yılanından gelmiyor, Monty Python adlı altı kişilik bir İngiliz komedi grubundan geliyor.
- Yaşlı değil, 1990 doğumlu bir programlama dili.
- Pay(th)ın diye okunur.
- Google başta olmak üzere NASA, HP gibi yerlerde çokça kullanılmaktadır. Baş geliştiricisi Google ve Dropbox'ta çalıştıktan sonra emekli oldu.

1.2 Temeller

1.2.1 python komutu

```
$ python
Python 2.7.7 (default, Aug 15 2014, 12:42:13)
[GCC 4.7.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Merhaba")
Merhaba
>>> print("Hello World!")
Hello World!
>>>
```

1.2.2 ipython komutu

1.2.3 Yorumlar (comments)

```
>>> #print("Hello World!")
...
>>>'''
print("Hello")
print("World")
'''
```

1.2.4 Veri tipleri

Aksi hali belirtilmediği sürece verinin tipi kullanıcı ilk atama yaptığı anda oluşturulur.

```
>>> a = 1
>>> b = 5
>>> print(a+b)
6
```

```
>>> a = "1"
>>> b = "5"
>>> print(a + b)
15
```

Görüldüğü üzere verilerin tiplerine python kendisi karar veriyor. İlk durumda int seçerken ikinci durumda string tipini seçiyor. Karışık veri tiplerinde ise şöyle çalışıyor.

```
>>> a = "0"
>>> b = 2
>>> print(int(a) + b)
2
```

Temelde üç adet basit veri tipi bulunmaktadır: int(), float(), string(). Bu fonksiyonlar veri tipleri arasındaki dönüşümü sağlamaktadır (casting).

```
>>> a = 67
>>> print(a)
67
>>> str(a)
'67'
>>> float(a)
67.0
>>> int(str(a))
67
```

1.2.5 Operatörler

Aritmetik operasyonlar diğer programlama dillerinden çok farklı değil.

```
>>> print (3 + 4)
7
>>> print (3 - 4)
-1
>>> print (3 * 4)
12
>>> print (3 / 4)
0
>>> print (3 % 2)
1
>>> print (3 ** 4) # C'deki 3^4
81
>>> print (3 // 4) # C'deki floor(3/4)
0
```

1.2.6 Atamalar ve Çoklu Atamalar

Atama operatörleri de benzeri şekilde çalışıyor.

```
>>> a = 0
>>> a+=2
>>> print (a)
2
>>> a -= 3
>>> print(a)
-1
>>> a *=a
>>> print(a)
1
```

python'un getirdiği bir özellik de çoklu ve karışık veri tipli atama yapılabilmesidir.

```
>>> f, g, h = 4, "t", 6.1
>>> print(f, g, h)
(4, 't', 6.1)
>>>
```

1.2.7 Akış Kontrolü

Aşağıdaki `if` bölümünü inceleyelim. Her `if` satırı ":" ile bitiyor. Altındaki, yani etki alanı (scope) bir paragraf girişi kadar içeriden başlıyor. Döngülerde, fonksiyonlarda ve bu gibi koşullu etki alanı işlemlerinde boşluklar çok önemlidir. Aynı anda hem okunabilirliği artırırken, hem de kullanıcıyı düzenli kod yazmaya mecbur bırakır.

```
>>> if a>= 22:
...     print("if")
... elif a >= 21:
...     print("elif")
... else:
...     print("else")
...
else
```

Boşluklara dikkat. Bu paragraflamanın standardı 4 boşluk karakteridir. Ama bu bir kural değildir.

1.2.8 Özel kelimeler

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

1.2.9 Sabit ve değişken veri tipleri (immutable vs mutable)

Sabit veri tipine (immutable) atanmış öğeler değiştirilemezler. Bunun yerine o veri tipinin yenisini oluşturuken değişikliği yapıp aynı öğenin üzerine yazmak gerekir. Bu veri tiplerinden ikisi `string` ve `tuple`dır.

```
>>> a = "Merhaba"
>>> print(a[3])
'h'
>>> a[3] = b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> print(a[3])
h
```

Görüldüğü üzere bir dizinin herhangi bir indisine ulaşmak için `[]` parantezleri kullanılıyor. Yukarıdaki örnekte bir string değişkeninin içini (3. indisindeki karakteri) değiştirmemize izin vermemektedir.

1.3 Diziler

1.3.1 Lists

C++'deki vector veri tipine benzetilebilir. Veriler 0'dan başlamak üzere indekslenir. Listeden veriler çıkarılabilir, değiştirilebilir ve sona yeni değerler eklenebilir. Liste oluşturmak için `[]` parantezleri kullanılır. `.append()` metodu ve `del()` fonksiyonu sırasıyla sona veri ekleme ve veri silmek için kullanılabilir.

```
>>> l = [ 'A', 'C', 'C', 'G', 'T']
>>> print(l)
['A', 'C', 'C', 'G', 'T']
>>> print(l[5])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> print(l[3])
G
>>> l[3] = 'K'
>>> print(l)
['A', 'C', 'C', 'K', 'T']
>>> l.append('A')
>>> print(l)
['A', 'C', 'C', 'K', 'T', 'A']
>>> del l[0]
>>> print(l)
['C', 'C', 'K', 'T', 'A']
>>> 'C' in l
True
>>> 'Y' in l
False
```

1.3.2 Tuples

Tuple dizi tipi list tipiyle aynı şekilde kullanılır. Ama sabit (immutable) bir veri tipidir. Tuple içindeki veriler değiştirilemez, çıkartılamaz ve sona yeni değerler eklenemez. Tuple oluşturmak için `()` parantezleri kullanılır.

```
>>> months = ('January','February','March','April','May','June',\
... 'July','August','September','October','November','  December')
>>> print(months)
('January', 'February', 'March', 'April', 'May', 'June', 'July', 'Au
gust', 'September', 'October', 'November', '  December')
>>> months[3] = 'Nisan'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

1.3.3 Sözlükler (dicts)

Adından da anlaşılacağı gibi sözlük gibi bir fihrist oluşturmaya yarayan bir dizi tipidir. Her satır iki değişkenden oluşur: Bir anahtar (key) ve bir de değer (value). List ve tuple dizi tiplerinden farklı olarak her değere bir numara ile değil anahtar verilerek ulaşılır. Dict oluştururken {} parantezleri kullanılır. Sabit veri tipi değildir. İstenilen değişiklik yapılabilir. C++'daki `map` dizi tipine benzetilebilir.

```
>>> #Make the phone book:
... phonebook = {'Andrew Parson':8806336, \
... 'Emily Everett':6784346, 'Peter Power':7658344, \
... 'Lewis Lame':1122345}
>>> print(phonebook)
{'Emily Everett': 6784346, 'Andrew Parson': 8806336, 'Lewis Lame':
1122345, 'Peter Power': 7658344}
>>> phonebook['Gingerbread Man'] = 1234567
>>> print(phonebook)
{'Emily Everett': 6784346, 'Andrew Parson': 8806336, 'Lewis Lame':
1122345, 'Peter Power': 7658344, 'Gingerbread Man': 1234567}
>>> del phonebook['Andrew Parson']
>>> print(phonebook)
{'Emily Everett': 6784346, 'Lewis Lame': 1122345, 'Peter Power': 765
8344, 'Gingerbread Man': 1234567}
```

```
>>> ages = {}
>>> ages['Ali'] = 15
>>> ages['Ahmet'] = 20
>>> ages['Ayse'] = 25
>>> print(ages)
{'Ahmet': 20, 'Ayse': 25, 'Ali': 15}
>>> if ages.has_key('Ali'):
...     print "Ali var ve", ages['Ali'], "yasinda."
... else:
...     print "Ali yok."
...
Ali var ve  15 yasinda
>>> print(ages.keys())
['Ahmet', 'Ayse', 'Ali']
>>> print(ages.values())
[20, 25, 15]
>>> len(ages)
3
>>> keys = ages.keys()
>>> values = ages.values()
>>> print(values)
[20, 25, 15]
>>> values.sort()
>>> print(values)
[15, 20, 25]
```

1.3.4 Strings

Değişik string kullanımları:

```
>>> 'spam eggs'  # Tek kesme isareti
'spam eggs'
>>> 'doesn\'t'  # Escape karakteri
```

```
"doesn't"
>>> "doesn't"  # ...ya da cift tirnak.
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

```
>>> print("First line.\nSecond line.")
First line.
Second line.
>>> print("""First
... Second
... Third""")
First
Second
Third
```

```
>>> print(3 * 'Merhaba')
MerhabaMerhabaMerhaba
>>> print('Py' 'thon')
Python
>>> print(pre 'thon')
  File "<stdin>", line 1
    print(pre 'thon')
                 ^
SyntaxError: invalid syntax
```

```
>>> word = "Python"
>>> len(word)
6
>>> print(word[4])
o
>>> print(word[-1])
n
>>> print(word[0:2])
Py
>>> print(word[0:4])
Pyth
>>> print(word[:4])
Pyth
>>> print(word[2:4])
th
>>> print(word[-2:])
on
```

1.3.5 Dizi tipleri arasında geçiş

```
>>> l = [5, 4, 1, 6, 7]
>>> print(l)
[5, 4, 1, 6, 7]
>>> t = tuple(l)
>>> print(t)
(5, 4, 1, 6, 7)
>>> l = list(t)
```

```
>>> print(l)
[5, 4, 1, 6, 7]
```

- `int(x [,base])`: Tamsayı tipine dönüştürür. `base`, `x` bir karakter katarı ise mod tabanını belirler.
- `long(x [,base])`: Uzun tamsayı tipine dönüştürür.
- `float(x)`: Kayan ondalık sayıya dönüştürür.
- `complex(real [,imag])`: Karmaşık sayı oluşturur.
- `str(x)`: Karakter dizisi gösterimine dönüştürür.
- `repr(x)`: Karakter dizisi ifadesine dönüştürür.
- `eval(str)`: Karakter dizisini hesaplar.
- `tuple(s)`: Tuple tipine dönüştürür.
- `list(s)`: Liste tipine dönüştürür.
- `set(s)`: Küme tipine dönüştürür.
- `dict(d)`: Sözlük oluşturur. `d`'nin (anahtar, değer) ikililerinden oluşan bir tuple olması gerekir.
- `frozenset(s)`: Donuk küme şekline dönüştürür.
- `chr(x)`: Karakter tipine dönüştürür.
- `unichr(x)`: Unicode karakter tipine dönüştürür.
- `ord(x)`: Tek karakteri ascii tam sayı değerine dönüştürür.
- `hex(x)`: Tamsayıyı onaltılık düzendeki ifadesine dönüştürür.
- `oct(x)`: Tamsayıyı sekizlik düzendeki ifadesine dönüştürür.

1.4 Dosyadan çağırma

python dilinde yazılmış bir kod parçasını python komut satırı yerine bir dosyadan çağırabiliriz. Bunun için ".py" uzantılı dosyalar kullanılmaktadır. "merhaba.py" adında bir dosya oluşturup bunu çalıştırmaya çalışalım. Aşağıdaki kod parçasını bir "merhaba.py" adlı bir dosya içine yazalım.

merhaba.py

```
1 #!/usr/bin/env python
2
3 print("Hello World!")
```

Sonra dosyamızı çalıştırılabilir hale getirip çalıştırıyoruz.

```
$ chmod +x merhaba.py # make the file executable
$ ./merhaba.py        # run the command
Hello World!
$
```

1.5 Hata izleme

Hatalar söz dizimi, tanımsız değişken, ulaşılamayan kaynak gibi bir çok sebeple ortaya çıkabilir. Aşağıdaki hata mesajını inceleyelim.

```
Traceback (most recent call last):
  File "test.py", line 25, in ?
    triangle()
  File "test.py", line 12, in triangle
    inc_total_height()
  File "test.py", line 8, in inc_total_height
    total_height = total_height + height
UnboundLocalError: local variable 'total_height' referenced before
assignment
```

Programlar hatanın olduğu yere kadar çalışır, varsa çıktılarını basar. python'da hata mesajları geriye dönük olarak komutun başlangıcından hatanın gerçek yerine doğru olan bütün yolu (Traceback) gösterir. Yukarıdaki örnekte çalışma "test.py" ile başlıyor, 25. satırında triangle() fonksiyonu çalıştırılıyor. triangle() fonksiyonunun 12. satırında inc_total_height() adlı fonksiyon çalıştırılıp, onun da 8. satırında hatayla karşılaşılıyor. Bu şekilde hatanın başlangıçtan sona kadar nerelerden geçtiğini anlayabiliyoruz.

1.6 Ödev I - Basit veritabanı oluşturma

- Komut çalıştırıldığında komut parametrelerinden ilki kullanıcıya kaç defa (N) soru soracağını belirtecektir: `sys.argv`
- Program kullanıcıya numara (ID), isim, soyisim ve yaş soracaktır, bu soruları N defa tekrarlayacaktır.
- Kullanıcı bilgileri tek satırda yukarıdaki sıraya göre girecek, program bunların arasındaki boşluklara göre yorumlayacaktır. İsim birden fazla olabilir. Soyisim tek kabul edilecek.
- Bilgiler doğru olarak girildiğinde ayrı değişkenlere atama yapılacak. Numaranın ve yaşın sayı olması, isimlerin de karakter olmaları kontrol edilmelidir.
- Toplanan bilgiler anahtarları numara olan sözlüklere yerleştirilecek. Değerler isim, soyisim ve yaştan oluşan tuple'a konulacak.
- Verilen bir numaranın sözlükte olup olmadığı kontrolü yapılacak, olması durumunda kullanıcı uyarılacaktır.

- Çıkarken numara sırasına göre bilgiler ekrana bastırılacaktır.

1.6.1 Github

1. Sayfanız altında yeni bir repository olacak. Repository adı `dagitik_2020` olacak.
2. Bu repository altında her ödev için ayrı bir dizin olacak.
3. Bu ödevin dizin ismi `odev01` şeklinde olacak.

1.6.2 Teslim edilecekler

1. Son teslim zamanı: **14.10.2020**- 13:00 (TSİ)
2. Bölüm 1.6.1 altındaki adımları tamamladıktan sonra adı `odev01.py` olan dosyanızı `odev01` altına yerleştireceksiniz.
3. Depolarınız gizli (private) olacak. Ayrıca görebilmeleri için hocalarınızı da (*philotuxo* ve *at-ay*) contributor olarak repository sayfası altına eklemeniz gerekmektedir.
4. Ödevi notlayabilmemiz için ödevinizi görebilmemiz gerekmektedir.

Bölüm 2

python'a giriş II - 21.10.2020

2.1 Fonksiyonlar

python'da fonksiyonlar daha önce tecrübe ettiğimiz diğer programlama dillerindeki işlevi görmektedir. Hem düzenli şekilde kod parçalarını bir arada tutmaya, hem de birden fazla kullanılacak bir kod grubunu tek satırda çağırmak için kullanılmaktadır.

python'da fonksiyon oluşturmak için "def" komutu kullanılmaktadır. Aşağıdaki betik (script) parçasında "someFunction" adında bir fonksiyon tanımlanıp çağırılıyor. Kapsama (scope) yine dikkat etmek gerekiyor, fonksiyon tanımı ":" ile biterken içindekileri belirtmek için 4 karakter paragraf içinden yazıyoruz. "print" işlemi fonksiyonun içinde yer alırken "someFunction()" çağrısı dışarıda duruyor.

someFunction.py

```
1 def someFunction():
2     print("boo")
3
4 someFunction()
```

çıktı

```
boo
```

2.1.1 Fonksiyon parametreleri

Fonksiyon parametreleri anlaşılacağı üzere () arasına yazılıyor.

someFunction.py

```
def someFunction(a, b):
    print(a+b)
```

```
someFunction(12,451)
```

çıktı

```
463
```

2.1.2 Fonksiyon kapsamı

Değişkenler global olarak tanımlanmadığı sürece global olarak kabul edilmezler. Fonksiyon kapsamında oluşturulmuş değişkenler aksi belirtilmediği sürece sadece kapsam içinde tanımlıdır.

someFunction.py

```
def someFunction():
    a = 10

someFunction()
print (a)
```

çıktı

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

someFunction.py

```
a = 10

def someFunction():
    print (a)

someFunction()
```

çıktı

```
10
```

2.2 Döngüler

Döngüler python'un diğer programlama dillerine kıyasla en güçlü olduğu öğelerinden biridir. Çok basit ve ve çok yönlüdürler.

2.2.1 for

Fonksiyonlar gibi döngü başlangıçlarında ":" kullanıyoruz ve içerisini belirtmek için satır içerisinden yazıyoruz.

```
for a in range(1,3):
    print(a)
```

çıktı

```
1
2
```

Yukarıdaki örnekte `range()` fonksiyonu kullanılıyor. Bu fonksiyon aslen bir tuple oluşturuyor ve `in` sözcüğü `a` değişkeninin bu tuple içindeki değerleri alıp almadığını sorgulamaya yarıyor. `for` ile birleştirdiğimizde `a` bütün tuple içindek değerleri sırasıyla alıyor.

Dikkat edilmesi gereken bir husus `range(a,b)` fonksiyonu `a`'dan `b`'ye kadar giden tam sayıları döndürürken `b` sayısını kümenin bitiş sayısı olarak alıp asıl tuple içine eklemez.

2.2.2 while

`while` döngüsü de diğer programlama dillerindeki karşılığı benzeri gibi çalışmaktadır. Belirli bir şartı yerine getirene kadar döngü çalışmaya devam eder.

```
a = 1
while a < 10:
    print (a)
    a+=1
```

çıktı

```
1
2
3
4
5
6
7
8
9
```

`while` döngülerine her zaman doğru döndürecek (1==1 gibi) şartlar yazıldığı zaman sonsuz döngüler elde edilebilir. Bu şekilde kullanımı dinleyiciler (listeners) ve hatta oyunlar yazmak için sıkça kullanılmaktadır. Ama istenilmeyen sonsuz döngülere düşmemek için dikkatli olmayı gerektirir.

2.2.3 do-while

python'da diğer programlama dillerinin aksine bu döngü bulunmamaktadır.

2.3 Diziler - Devam

```
myString = ""
print (type(myString))
```

çıktı

```
<class 'str'>
```

`type()` değişkenin veri tipini döndürdüğü için hata ayıklama için kullanılan çok önemli bir fonksiyondur.

2.3.1 String

Aşağıda string özelinde kullanılabilen metodlar yer almaktadır.

- `.count(x)` - `stringVar` içindeki `x` sayısını getirir.
- `.find(x)` - `x` karakterinin konumunu getirir.
- `.lower()` - `stringVar`'ın küçük harfli halini getirir.
- `.upper()` - `stringVar`'ın büyük harfli halini getirir.
- `.replace(a,b)` - Bütün `a`'ları `b` ile değiştirir.
- `.strip()` - Baştaki ve sondaki boşluk karakterlerini temizler.

2.3.2 List

- `.append(a)` - `a`'yı listenin sonuna ekler.
- `.count(x)` - `x`'leri sayar.
- `.index(x)` - `x` karakterinin indisini getirir.
- `.insert(y,x)` - `y` konumuna `x` yerleştirir.
- `.pop()` - En sondaki elemanı döndürür ve siler.
- `.remove(x)` - `x`'i bulur ve siler.
- `.reverse()` - Listeyi ters çevirir.
- `.sort()` - Listeyi alfabetik veya sayısal olarak artan olarak sıralar.

2.3.3 Dict

Dict yapısını kullanırken anahtarlar tek olmalıdır. Aynı anahtara birden fazla değer eşleştirilemez, yani multimap olarak çalışmazlar.

```
>>> D = {'ali': 12, 'ahmet' : 15}
>>> print(D['ahmet'])
15
>>> print(D['ali'])
12
>>> D['ahmet'] = 25
>>> print(D['ahmet'])
25
```

2.3.4 Çıktı biçimlendirmesi (Formatting)

python'da çıktı biçimlendirmesi biraz değişiktir. Ekrana bastırılacak karakter dizisiyle verilen değişkenler arasına '%' konur. C'dekine benzer olarak '%' ayrıca kayan sayıların hassasiyetlerini belirlemede de kullanılmaktadır.

```
print('Fiyat: %f' % 123.44)
print('Fiyat: %.2f' % 123.444)
print('Fiyat: %.2f TL/%d adet ve %.2f TL/%d adet' % (123.444, 1, 233.44, 2))
```

çıktı

```
Fiyat: 123.440000
Fiyat: 123.44
Fiyat: 123.44 TL/1 adet ve 233.44 TL/2 adet
```

İlk '%'yi takip eden `f`, `float` kısaltması ve kayan noktalı sayının gösterilme biçiminin nasıl olması gerektiğini söylüyor. Bununla beraber bu sayının basılacak karakter dizisinde nerede olacağını gösteriyor. İkinci '%' ise katar içinde gösterilecek sayı değişkenlerini çağırıyor. python sayı katarını soldan itibaren okumaya başlar. İlk karşılaştığı '%' yerine ilk değişkeni ve ardından gelenler için sırasıyla biçimlendirmeye uyarak yerleştirir. Birden fazla sayı kullanılacağı zaman dışarıdaki '%' sonrasında tuple kullanılır.

Karakter katarları da sayı katarları gibi biçimlendirilebilir. Aşağıdaki örnekte bunu görebiliriz.

```
a ="abcdefghijklmnopqrstuvwxyz"
print('%.20s' % a)
```

çıktı

```
abcdefghijklmnopqrst
```

2.4 Hata yakalama (Exceptions)

Bir çok değişken ve kaynakla programlama yaparken hataları yakalama bir ihtiyaç olur. Veri tipi içeren basit değişkenler dışında, sistem süreçleri, sistem

girdisi gibi kontrol edilemeyen kaynaklar olabilir. Bazı durumlarda ise koddan ne çıkacağını tahmin bile edemeyiz. Her şeyi kontrol edemeyeceğimizi bilerek bu durumlarda hata durumlarına karşı önlem alıp, python'nun istenmeyen bir çıkış yapmasını engellemeliyiz. Bunun için de aşağıdaki gibi yazılımsal tedbirler almalıyız.

Kontrol edilemeyen basit bir durumu canlandıralım. Veri tipini bilmediğimiz bir değişkenle işlem yapmaya çalıştığımız bir durum olsun.

```
var1 = '1'
var1 = var1 + 1
```

çıktı

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

Yukarıdaki durumda yazdığımız kod parçası şekildeki gibi çıkacaktır. Karakter dizisi ile tamsayıyı toplayamayız. Olası bir hata durumunu yakalamak istersek bunu şöyle yazardık.

```
var1 = '1'
try:
    var1 = var1 + 1
except:
    print(var1, " is not a number")
print(var1)
```

çıktı

```
('1', ' is not a number')
1
```

Çalışma şekli de şöyledir. Çalıştırmak istediğimiz kod parçasını `try` içine koyarız. Bu kod çalıştırılır. Ancak bir problemle karşılaşıldığı bir durumda python `except` bloğu içindekini çalıştıracağını bilir. Şekildeki gibi hata durumu yaşandığı zaman programdan çıkmadan hata durumlarını düzenliyoruz.

Yukarıda yakaladığımız hatayı daha düzgün bir şekilde şöyle yapabilirdik.

```
var1 = '1'
try:
    var2 = var1 + 1
except:
    var2 = int(var1) + 1
print(var2)
```

çıktı

```
2
```

Bu daha tecrübeli bir programcının çalışma şekli, hatanın nasıl bir durumdan kaynaklanabileceğini tahmin edip ona göre önlem alıyor ve yazdığı programın önünde sonunda istediği şekilde çalışmasını sağlıyor. Bu durum başka nasıl yapılabilirdi? (`type`)

2.5 I/O işlemleri

2.5.1 Dosya okuma

readme.txt

```
Ben bir deneme dosyasiyim.
Bu ikinci satir.
Bu da ucuncu satir.
```

Zaman zaman programcı dosya sisteminden bir dosyayı okuyup, içindekileri işleme ihtiyacı duyacaktır. Bu bir veri dosyası veya ayar dosyası olabilir. python'da okuma işini de kolaylaştıran araçlar bulunmaktadır.

```
f = open("readme.txt", "r")
```

python'a hangi dosyayı okuması gerektiğini söylememiz gerekiyor. Bu durumda `readme.txt`. `open` metodu da dosyayı okumayı (veya bağlantı açmayı) sağlayan metoddur. 'r' dosyanın okumak için (**r**ead) açılacağını gösteren parametredir. Dosyada değişiklik yapmak isteseydik ise onu 'w' (**w**rite) ile açacaktık. Dosyayı açtıktan sonra da onu dosya tanımlayıcısı denilen bir değişkene atarız. Bu durumda `f` oluyor. Dosya üzerinde yapılan bundan sonraki işlemlerde sadece bu tanımlayıcı kullanılır.

Temel dosya okuma metodları aşağıdaki gibidir.

- `file.read(n)` : Açılmış dosyadan `n` tane karakteri okur, `n` verilmezse bütün dosyayı okur.
- `file.readline(n)` : Parametre verildiğinde `file.read(n)` gibi çalışır. Ama parametre verilmezse, bütün bir satırı okur.

```
f = open("readme.txt","r")
print(f.read(1))
print(f.read())
```

çıktı

```
B
en bir deneme dosyasiyim.
Bu ikinci satir.
Bu da ucuncu satir.
```

Dosyadan bir parça okunduğunda python kaldığı yerden okumaya devam eder. Yukarıdaki örnekte tek bir karakter okuduktan sonra, parametre verilmediği için ikinci karakterden sonuna kadar okur. Başka bir deyişle dosyanın iteratörünü kendi içerisinde saklar ve bir sonraki işlemde kaldığı yerden devam eder.

```
f = open("readme.txt","r")
print(f.readline())
print(f.readline())
```

çıktı

```
Ben bir deneme dosyasiyim.
Bu ikinci satir.
```

Benzeri şekilde `readline()` metodu da dosyada kaldığı yerden devam eder. Parametresiz çalıştırıldığında yukarıdaki gibi tam satırları okur.

```
f = open("readme.txt","r")
myList = []
for line in f:
    myList.append(line)
print(myList)
```

çıktı

```
['Ben bir deneme dosyasiyim.\n', 'Bu ikinci satir.\n', 'Bu da ucuncu satir.\n']
```

Yukarıdaki örnekte dosya içerisindeki her satırı bir listenin bir elemanı olarak kaydediyoruz. `for` döngüsü şartındaki `in` operatörü, `f`'den her seferinde bir satır okunmasını sağlıyor. Ayrıca python biçimi tamamen korumaya çalıştığı için satır sonlarındaki `\n` (yeni satır) karakterini de saklamaktadır. Daha sonra yapılacak işlemlerde bu fazladan gibi görünen karakterler istendiği gibi atılabilir (mesela `str.strip()` metoduyla).

Dosyayla işimiz bittiğinde dosyayı kapatmamız gerekiyor.

```
f.close()
print(f.read())
```

çıktı

```
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
ValueError: I/O operation on closed file
```

2.5.2 Dosya yazma

Yazma işlemine bir uyarı ile başlıyoruz. Dosyaya yazma işlemi yapmayacaksak kesinlikle 'w' ile açmamamız gerekiyor. Yanlış bir harekette silmek istemediğimiz verileri kaybedebiliriz. 'w' parametresini verdiğimiz anda yeni bir dosya açılıp varsa daha öncekinin üzerine yazılır.

```
f = open("readme.txt","w")
f.close()
```

Yukarıdaki örnek hiçbir şey yapmıyor gibi görünse de `readme.txt` dosyamızın içindeki verilerimizi yok etmektedir. Bu çoğu zaman kaçınmamız gereken bir davranıştır.

Yazarak dosyamızı yeniden oluşturalım:

```
f = open("readme.txt","w")
f.write('Ben bir deneme dosyasiyim.')
f.write('Bu ikinci satir.')
f.write('Bu da ucuncu satir.')
f.close()
```

Burada farketmemiz gereken nokta dosyaya yazdığımız karakter katarlarını ardı ardına sıralanması. Sonda yeni satır karakteri (`\n`) olmaması yeni satır oluşturulmayacağını gösteriyor. Bu yüzden dosya içerisi aslında şöyle oluyor.

readme.txt

```
Ben bir deneme dosyasiyim.Bu ikinci satir.Bu da ucuncu satir.
```

Satır satır yazmak istediğimiz bir karakter dizisinin satır başlarına yeni satır karakterleri (`\n`) eklememiz gerekiyor.

```
f.write('Ben bir deneme dosyasiyim.\n')
```

Bir dosyayı sadece okumak için açıyorsak 'r', yazmak için 'w' parametrelerini kullanmamız gerekiyor. Ancak 'w' kullandığımız zaman dosya içindeki her şeyin de silindiğini görmüştük. Bu iki parametreyle dosyaya bir şey eklemek pratik olmayacaktır. Bunun yerine sonuna ekle ('append') anlamına gelen 'a' parametresi kullanılır.

```
f = open("readme.txt","a")
f.write('Dorduncu satir.')
f.close()
```

2.6 Sınıflar ya da Nesne şablonları (Classes)

Bu bölümde öğrencinin Nesne Yönelimli Programlama yapısına aşina olduğu farzediliyor. Hatta öğrencinin Java ya da C# programlaması yaptığı farzediliyor.

ClassOne.py

```
class Calculator(object):
    #define class to simulate a simple calculator
    def __init__ (self):
        #start with zero
        self.current = 0
```

```
    def add(self, amount):
        #add number to current
        self.current += amount
    def getCurrent(self):
        return self.current
```

Yukarıdaki örnek adından anlaşılacağı üzere çok basit bir hesap makinası örneği. Kurduğumuz `Calculator` sınıfı `object` denilen başka bir sınıftan türetiliyor (*inheritance*). Buradaki `object` en üst sınıftır. Başka bir sınıf kullanmayacağımız zaman sadece bu sınıf kullanılır.

`def __init__ (self):`, başlatma (initialize) fonksiyonudur, sınıftan yeni bir nesne oluşturur. Nesne değişkenleri (attributes) burada oluşturulur. Parametre olarak kullanılan `self` nesneye ait olan değişkenlerin bu fonksiyon içinde kullanılması için gerekli bir parametre (sınıf değil).

Sınıfnun içine eklenen `add` metodu da benzeri şekilde (diğer metodlar gibi) `self` parametresini alır. Anlaşılacağı üzere `self` özel bir parametre, hatta nesne tanımlayıcısıdır. C++'daki 'this'e benzetilebilir. Metodlara bunu veriyor olmamız, fonsiyon çalışırken bunun hangi nesneye ait olduğunu bilmesi için gereklidir. Bu fonksiyona, `add`, özel olarak da `self.current` değişkeninin bu nesne içinde tanımlanmış bir değişken olduğunu bilir ve onu verilen diğer parametre kadar artırır.

Nesne değişkenlerini tekrar okuyabilmek için onları nasıl okuyacağımızı bilmemiz gerekmektedir. Diğer programlama dillerindeki nesne değişkenlerine ulaşmak için kullanılan metodları (accessor) burda da tanımlamak gerekir. Nesnenin o andaki `current` değişkenini dışarıdan okuyabilmek için `getCurrent()` diye bir metod tanımlarız.

Aşağıdaki örnek kod parçasında yukarıda tanımlanmış sınıfı kullanarak bir nesne oluşturuluyor ve tanımlanmış metodlarını kullanarak üzerinde işlemler yapılıyor.

deneme.py

```
1 from ClassOne import *
2
3 myBuddy = Calculator()
4 myBuddy.add(2)
5 print(myBuddy.getCurrent())
6 myBuddy.add(7)
7 print(myBuddy.getCurrent())
```

Yukarıdaki satırları farklı bir "deneme.py" dosyasına kaydedip "ClassOne.py" ile aynı dizinde çalıştırdığımız zaman aşağıdaki çıktıyı alırız.

çıktı

```
2
9
```

`deneme.py` dosyasını satır satır anlatalım. 1. satırdaki `from ClassOne` hangi dosyadan okuma yapacağımızı gösterir. `import *` ise bu dosyadan hangi sınıfların

kullanılacağını gösteriyor. '*', bütün sınıfları kullanmak istediğimize işaret eder. Bizim durumumuzda sadece bir tane sınıf (`Calculator`) bulunmaktadır. Satır 3'te `myBuddy` adında yeni bir nesne oluşturuyoruz. Bu nesneyi de `Calculator` sınıfına atayarak aslında `Calculator` sınıfından yeni bir nesne oluşturmuş oluyoruz. Sınıfımızda iki tane metodumuz vardı: `.add()` ve `.getCurrent()`. Dolayısıyla nesnemizle bu metodları kullanabiliriz. Ayrıca nesnemiz oluşturulurken `__init__()` metodu da çağırılacağı için nesne değişkeni olan `current`'a da 0 atanmış durumda olacaktır. 4. satırda `myBuddy.add(2)` çağrısıyla bu değere $(0 + 2 =)$ 2, 6'dakiyle de $(2 + 7 =)$ 9 ataması yapılacaktır.

Aslında bütün yukarıda gördüğümüz tuple, string, list, dict gibi değişkenler de sınıflardan oluşmaktadır. Sıkça kullanıldıkları için özel olarak başta sanki veri tipleri olarak verilmektedir.

2.6.1 Object-Oriented Terminology

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables aren't used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Function overloading: The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects (arguments) involved.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.
- Inheritance : The transfer of the characteristics of a class to other classes that are derived from it.
- Instance: An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- Instantiation : The creation of an instance of a class.
- Method : A special kind of function that is defined in a class definition.
- Object : A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
- Operator overloading: The assignment of more than one function to a particular operator.

Daha ileri sınıf örnekleri için: Python Object Oriented.

2.7 Ödev II

Bu ödevde amacımız, verilen veri dosyasına göre bir havayolu şirketinde kazanılan mil puanlarının başka bir havayolu şirketinin mil puanlarına çevrilip çevrilmeyeceğini kontrol etmektedir. Veri dosyasındaki satır başlarındaki havayolu şirketinin partner şirketleri satırın devamında yer almaktadır. Veri dosyası bir ağ yapısını temsil edip, puanların birden fazla aşamada çevrilmesi mümkündür. Başka bir deyişle; amaç, bir havayolu şirketinden başka bir havayolu şirketine bu veri dosyasına göre bir yol (path) var mı sorusunu cevaplamaktır, varsa bu yolu ekrana bastırmaktır.

Dosyaların kendilerini bu linklerden de indirebilirsiniz: Havayolu problemi java kodu ve Partner ağı veri dosyası.

2.7.1 Veri dosyası - Partnerlik ilişkisi

airlines.txt

```
Delta,Air Canada,Aero Mexico,Ocean Air
United,Aria,Lufthansa,Ocean Air,Quantas,British Airways
Northwest,Air Alaska,BMI,Avolar,EVA Air
Canjet,Girjet
Air Canada,Aero Mexico,Delta,Air Alaska
Aero Mexico,Delta,Air Canada,British Airways
Ocean Air,Delta,United,Quantas,Avolar
Aria,United,Lufthansa
Lufthansa,United,Aria,EVA Air
Quantas,United,Ocean Air,AlohaAir
BMI,Northwest
Maxair,Southwest,Girjet
Girjet,Southwest,Canjet,Maxair
British Airways,United,Aero Mexico
Air Alaska,Northwest,Air Canada
Avolar,Northwest,Ocean Air
EVA Air,Northwest,Lufthansa
Southwest,Girjet,Maxair
AlohaAir,Quantas
```

2.7.2 İstenen ve Çıktı

Sizden istenen bu problemi python'la çözmektir. Bunun için önce `airlines.txt` dosyasını dışarıdan okuyacaksınız. Okurken işinize en çok yarayacak veri tipine aktaracaksınız. Bir havayolu şirketinden başka birine ulaşılabiliyor musunuz diye arama yapmak gerekecek. Sonuç olarak, rastgele havayolu şirketi çiftleri parametre olarak verildiğinde, yazacağınız python programının bunların arasında partnerlik ilişkisi olup olmadığını döndürebilmesi varsa yolu ekrana basması gerekiyor.

2.7.3 Teslim edilecekler

1. Son teslim zamanı: **21.10.2020**- 13:00 (TSİ)

2. Ödevinizi github deponuzun `odev02` dizinin altına `odev02.py` ismiyle göndereceksiniz.

3. Ödevi notlayabilmemiz için ödevinizi görebilmemiz gerekmektedir.

2.8 Ödev III - Veri istatistikleri

İç ortamlarda konumlama yapabilmek için değişik veri tipleri (modes) kullanılmaktadır. Kamera, Ladar, Lidar, Infrared ve radyo frekansı sinyalleri bunlardan bir kaçıdır. Dış ortamlarda Küresel Konumlama Sistemi (GPS) ile metre hassasiyetine kadar indirilebilmiş genel geçer bir sistem kullanılmaya başlanmışken, iç ortamlarda yüksek hassasiyette ve ucuz aynı şekilde bir çözüm henüz bulunmamaktadır.

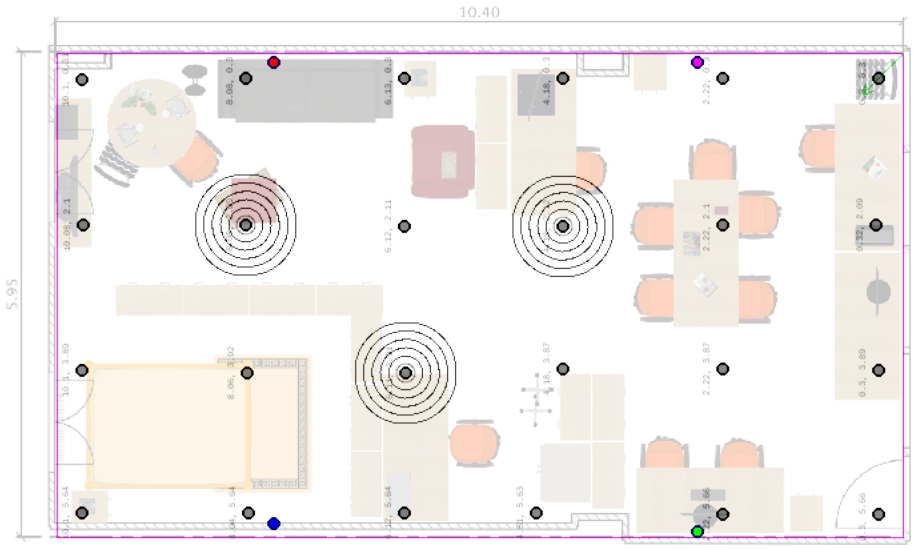
Son yıllarda kullanılan aygıtların ucuzluğundan dolayı radyo frekans sinyalleri çok öne çıkmaktadır. Bu yöntemlerde genellikle indirgenmiş yayın sinyal şiddeti (reduced signal strength indicatior) kullanılarak verici ve alıcı arasındaki uzaklık veya doğrudan konum tahminlemesi yapılmaktadır. Sinyal olarak önceleri kablosuz haberleşmede kullanılan Wi-Fi altyapısı kullanılırken, 2010 yılında Bluetooth 4.0'ın devreye girmesiyle iç ortam konumlama sistemleri için Bluetooth Low-Energy (BLE) aygıtları yoğun olarak kullanılmaya başladı. Bu sistemlerde genellikle yıllarca aynı pille çalışabilen BLE paketi yayınlayıcılar (beacon) ve BLE paketlerini yakalayabilen herhangi bir aygıt (akıllı telefon veya bilgisayar) kullanılmaktadır. Vericiler sadece pille çalışıyor ve görece ucuz olmalarından dolayı genellikle sabit olarak (yerleştir ve unut) düşünülürken, algılayıcılar ise genellikle hareketli düşünülüp bunların konumunun tayin edilmesi problemi çözülmeye çalışılmaktadır.

Ödevimizde yukarıda anlatılan genel yöntemin tersi bir durum vardır: Ortamda konumları sabit algılayıcılar ve hareketli vericiler bulunmaktadır. Amaç vericilerin konumlarını tahminlemekdir. Parmak izi (fingerprint) toplaması da denilen bu saha analizinde, konumu tahminlenecek aygıt bir çok noktada belirli bir süre tutularak bu noktalardaki veriler kaydedilmektedir. Daha sonra bu verilerden oluşturulmuş radyo frekans haritası gibi değişik istatistikler canlı konumlamada kullanılmaktadır.

Ödevimiz, benzeri bir konumlama sisteminin saha analizinin bir kısmını içermektedir. İki adet BLE vericisi Şekil 2.1'de gri ile işaretlenmiş konumlara 15 dakikalığına bırakılarak yayınladıkları mesajların renkli noktalarla işaretlenmiş algılayıcılar tarafından yakalanması ve bu paketlerin RSSI değerlerinin kaydedilmesi sağlanmıştır. Her konumda toplanan veriler farklı dosyalar olarak kaydedilmiştir.

Bu dosyalar ilgili github dizinlerinizin içine uzantısı ".mbd" ile biten bir dosya olarak koyuldu. Her öğrencide sadece bir tane dosya bulunmaktadır. Aslında bir metin dosyası olan bu dosyalar virgülle ayrılmış sütunlardan oluşmaktadır. Bu sütunlar aşağıdaki verileri ifade etmektedir:

```
<timeStamp>,<sensor_mac>,<transmitter_mac>,<rssi>
```



Şekil 2.1: Bir laboratuvar ortamının parmak izi konumları. Renkli daireler algılayıcı konumlarını, gri daireler ise vericilerin saha analizi verisi toplamak için koyulduğu geçici konumları göstermektedir.

Örnek

```
...
1513241728.688545942,001583e5a5bd,e78f135624ce,-69
1513241728.645185947,001583e5a3c0,f963ea9bb3ea,-63
1513241728.686600923,001583e5b269,e78f135624ce,-62
1513241728.687197923,001583e5a3c0,e78f135624ce,-70
1513241729.104202985,001583e5a3c0,f963ea9bb3ea,-66
1513241729.104568004,001583e5b269,f963ea9bb3ea,-65
1513241729.104199886,001a7dda710b,f963ea9bb3ea,-56
1513241729.106705904,001583e5a5bd,f963ea9bb3ea,-71
1513241729.139194965,001583e5a3c0,e78f135624ce,-69
...
```

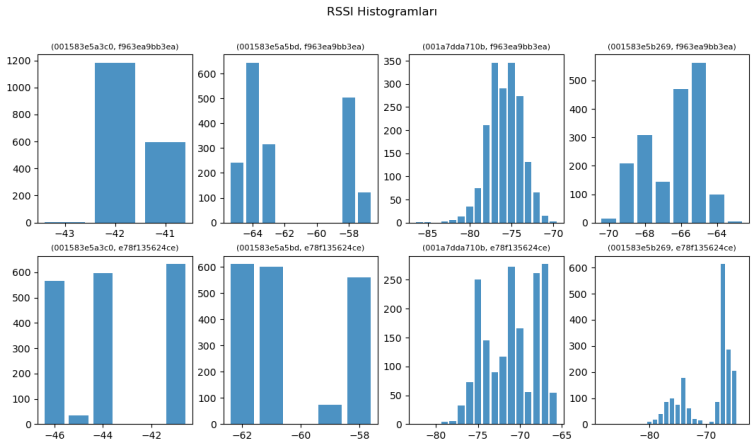
Bu ödevde sizden detayları ilerleyen bölümlerde anlatılan aşağıdaki istatistiksel bilgileri ve ilgili görselleri oluşturmanız istenmektedir:

1. Her (`sensor_mac`, `transmitter_mac`) çiftine göre RSSI değerlerinin dağılımı
2. Her (`sensor_mac`, `transmitter_mac`) çiftine göre anlık yayın frekanslarının değişimi
3. Her (`sensor_mac`, `transmitter_mac`) çiftine göre anlık yayın frekanslarının dağılımı

2.8.1 RSSI dağılımları (histogramları)

RSSI dağılımlarını hesaplayabilmek için hangi RSSI değerinden kaç tane olduğunu saymak gerekecektir. Adım adım yapılması gerekenler:

1. Sayılacak verilerin en yüksek ve en düşük değerleri belirlenecek,
2. Bu değerleri içerecek şekilde bir adımlı (step) sıralı bir sayı kümesi çıkarılacak, (`range` veya `numpy.arange` ile)
3. Aynı büyüklükte sıfırlardan oluşmuş başka bir küme oluşturulacak,
4. Veri dosyası satır satır okundukça RSSI değerleri sayılacak: 2 numarada oluşturulan kümedeki sayıların karşılığına gelen 3 numarada oluşturulan küme elemanları bir artırlacak,
5. Yukarıdaki işlemler her bir (`sensor_mac`, `transmitter_mac`) çiftine göre ayrı ayrı yapılacak.
6. En son olarak bu veriler `matplotlib.pyplot` altındaki `bar()` fonksiyonu ile görsellenecek. Örnek grafikler aşağıdaki gibidir.
7. `hist()` fonksiyonu kullanılmayacak.



Şekil 2.2: Örnek histogramlar

2.8.2 Anlık yayın frekansı değişimi ve dağılımı

Bulmak istediğimiz diğer istatistik, anlık yayın frekansları (momentary emission frequencies), yani belirli bir anda yayınlayıcının saniyede kaç tane mesaj yayınladığını bilmek istiyoruz.

Anlık yayın frekanslarını ard arda gelen birden fazla veriyi kullanarak bulacağız. Her adımda w büyüklüğünde bir pencerede, yani w tane ard arda gelen veri noktasının zaman değerlerini kullanarak, T_t anındaki anlık frekansı hesaplayacağız.

$$f_t = \frac{w}{T_t - T_{t-w}}$$

$w = 100$ olarak seçilecek.

Verileri saymak için uygulanması gereken adımlar:

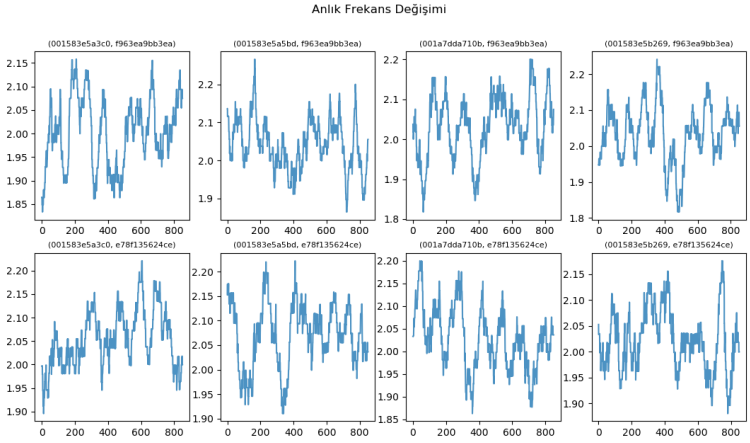
1. Anahtarları (`sensor_mac`, `transmitter_mac`) çiftleri olan bir (ya da iki) dictionary oluşturulacak.
2. Her (`sensor_mac`, `transmitter_mac`) çifti için iki liste (`list()`) açılacak. Bu listelerden ilki ham veri ZAMANının (timestamp) PENCERESİni tutmak için, diğeri de anlık yayın FREKANSlarını tutmak için kullanılacak.
3. w sayısı kadar eleman oluncaya kadar ilgili çiftin perceresine verilerin zamanları eklenecek.
4. w tane zaman oluştuğunda perceredeki son zamanla (T_t) ilk zaman (T_{t-w}), yukarıdaki formüle koyulup frekans (f_t) hesaplanacak ve bu frekans ikinci listeye, yani frekans listesine eklenecek.
5. Bundan sonra pencereye yeni bir eleman ekleneceği zaman baştaki eleman atılacak ve yeni eleman sona eklenecek. Böylelikle penceredeki eleman sayısı her zaman w olarak kalacak.
6. Her yeni eleman eklendiğinde ise 4.'teki hesaplama işlemi tekrarlanacak ve frekans verilerine eklenecek (`append`).

Bu verilerden iki farklı grafik görsellemek istiyoruz: Anlık frekans değişimleri ve anlık frekans dağılımları.

Anlık frekans değişimleri için `matplotlib.pyplot` modülünden `plot()` fonksiyonunun kullanarak bütün frekans verilerini görselleyeceğiz. (bkz. Şekil 2.3).

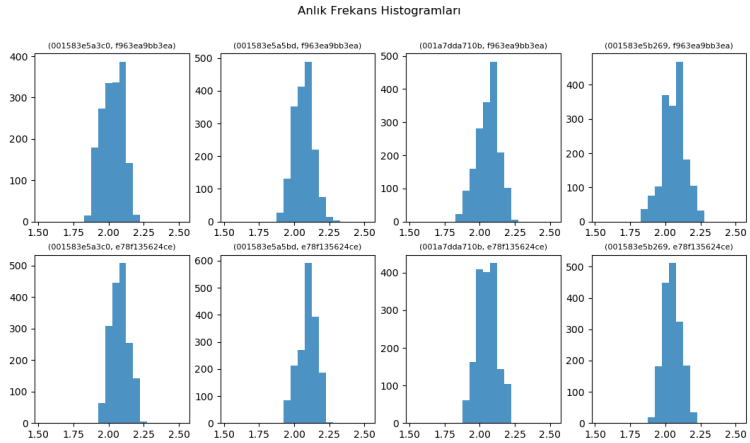
Anlık frekans dağılımları için ise RSSI dağılımlarındaki gibi verileri sayacağız.

1. Veriler reel sayılardan oluştuğu için hangi veriden kaç tane olduğunu saymak yerine değer aralıkları belirlenecek ve bunların arasına düşen veriler sayılacak.
2. Yazacağınız programda, 1.5 - 2.5 aralığını $b = 0.05$ boyutlarında pencerelere bölüp, bu aralıklara düşen veri sayılarını hesaplatılacak.
3. RSSI verilerinde olduğu gibi `matplotlib` modülünü kullanarak bar grafiği şeklinde ekrana bastırılacak (bkz. Şekil 2.4).



Şekil 2.3: Örnek değişim grafikleri

4. `hist()` fonksiyonu kullanılmayacak.



Şekil 2.4: Örnek histogramlar

2.8.3 Teslim edilecekler

1. Son teslim zamanı: **01.11.2020** - 22:00 (TSİ)
2. Ödevinizi github deponuzun `odev03` dizinin altına `odev03.py` ismiyle tek bir betik halinde göndereceksiniz.
3. Programınız grafikleri oluşturabilir ve gösterebilir olacak. Her grafik penceresi bir bölümün çıktılarını içerecek ve her pencere içinde örneklerde verildiği gibi 8 tane grafik olacak (2 verici $\times$ 4 algılayıcı). Vericiler satır,

algılayıcılar sütunları ifade ediyor olacak. Bunu yapabilmeniz için `subplots` veya `subplot` gibi bir görselleyici kullanmanız gerekecektir.

4. Grafikleri ekrana bastırmak için kullanılacak `show()` fonksiyonunun blocking (kesici) bir fonksiyonu olduğunun farkındayız. Bir grafik ekranda dururken diğer grafik görülmeyebilir. Bir grafiği kapadıktan sonra diğeri görüntülenebilir. 3 grafik penceresinin de ekranda aynı anda görünmesi şart değil.

5. Grafikleri ayrıca aynı dizin altında `odev03/figures/` diye başka bir dizine png formatında koyacaksınız. Grafikleri içeren png dosyalarının isimleri sırasıyla `grafik01.png`,
`grafik02.png`,
`grafik03.png`
olacak. Bunları grafikleri elde ettiğinizde elle (`matplotlib`'deki save tuşu) kaydedebilirsiniz.

6. Ödevi notlayabilmemiz için ödevinizi görebilmemiz gerekmektedir.

Bölüm 3

İleri python uygulamaları - 04.11.2020

3.1 İş parçacıkları (Threads)

Threads ya da "iş parçacıkları" bir sürecin (process) iş yapan birimleridir. Bir süreçte en az bir adet iş parçacığının olması gerekir. Buna ana iş parçacığı ya da "main thread"denir ve bu parçacık sonlandığında uygulamanın sonlaması beklenir. İş parçacıkları işletim sistemi tarafından önceliklerine göre belirli bir süre çalıştırılıp, dondurulurlar. Her bir parçacık için "time slice" denilen bir süre ayrılır. Bu işlem ve parçacıklar arasındaki geçiş çok hızlı gerçekleştiği için biz birçok uygulamanın aslında aynı anda iş yaptığını düşünürüz. Fakat aynı anda yapılan iş sayısı elimizdeki işlemci çekirdekleri sayısı kadardır.

İş parçacığı kullanmak, birçok programı eş zamanlı çalıştırmaya benzer. Ancak bununla beraber iş parçacıklarının sunucu ve istemciler özelinde aşağıdaki yararları vardır:

- Bir süreç (process) içinde bir çok parçacık ana parçacıkla aynı veri alanını paylaşır. Dolayısıyla ayrı süreçler olmasına kıyasla aralarındaki bilgi akışı ve haberleşme daha kolay olur.
- Bir kaynağın (dosya, soket vs.) tek bir parçacık tarafından kontrol edildiği bir senaryoda başka parçacıkların kaynağa ulaşması bu ortak bellek aracılığıyla kolay ve kontrollü olur.
- İş parçacıkları zaman zaman hafif süreçler olarak da anılırlar. Bellek bakımından fazla ek yük istemezler, kaynak bakımından süreçlerden veya çoklu süreçlerden daha ucuzdurlar.

Aşağıdaki unix komut ve parametreleriyle bir süreç altında çalışan iş parçacıklarını görebilirsiniz.

```
ps -T -p <pid>
```

```
top -H -p <pid>
```

3.1.1 `Threading` modülü

Modern Python3 `Threading` modülü üst seviye bir iş parçacığın desteği sağlamaktadır. Aşağıda `threading` modülünün metodları listelenmektedir:

- `threading.activeCount()`: Çalışan iş parçacığı (`thread`) nesnelerinin sayısını döndürür.
- `threading.currentThread()`: Çağıranın kontrolünde olan `thread` nesnelerinin sayısını döndürür.
- `threading.enumerate()`: Çalışmakta olan `thread` nesnelerinin listesini döndürür.

Bu metodların yanında `Threading` modülü bu derste çok kullanacağımız `Thread` sınıfını içerir. Aşağıdakiler de bu sınıfın metodları:

- `run()`: Bir iş parçacığının giriş noktası.
- `start()`: `run()` metodunu kullarak iş parçacığı başlatma.
- `join([time])`: İş parçacığının sonlanmasını beklemek için kullanılır.
- `isAlive()`: Bir iş parçacığı hala çalışıyor mu diye kontrol eder.
- `getName()`: İş parçacağının ismini döndürür.
- `setName()`: İş parçacağının ismini belirler.

3.1.2 `Threading` modülünü kullanarak yeni iş parçacığı oluşturma

Yeni bir iş parçacığı oluşturmak için `Thread` sınıfının mirasçısı olan yeni bir alt sınıf (subclass) oluşturmak gereklidir. Yeni parametreler eklemek için `__init__(self, [,args])` metodununun üzerine yenisi (override) yazılır. Benzeri şekilde `run()` metodu, iş parçacığının ne yapacağını içerecek şekilde üzerine yazılır. Alt sınıf oluşturulduğunda, bundan yeni bir nesne üretilir ve iş parçacığını başlatmak için `start()` metodu çağrılır. Bu metod `run()` metodunu çağıracaktır.

threading_example.py

```
1 #!/usr/bin/env python3
2
```

```
import threading
import time

exitFlag = 0

class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print ("Starting", self.name)
        print_time(self.name, self.counter, 5)
        print ("Exiting", self.name)

def print_time(threadName, delay, counter):
    while counter:
        if exitFlag:
            thread.exit()
        time.sleep(delay)
        print ("%s: %s" % (threadName, time.ctime(time.time())))
        counter -= 1

# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()

# Wait for all threads to complete
thread1.join()
thread2.join()
print("Exiting Main Thread")
```

çıktı

```
Starting Thread-1
Exiting Main Thread
Starting Thread-2
Thread-1: Mon Sep 15 12:24:05 2014
Thread-2: Mon Sep 15 12:24:06 2014
Thread-1: Mon Sep 15 12:24:06 2014
Thread-1: Mon Sep 15 12:24:07 2014
Thread-2: Mon Sep 15 12:24:08 2014
Thread-1: Mon Sep 15 12:24:08 2014
Thread-1: Mon Sep 15 12:24:09 2014
Exiting Thread-1
Thread-2: Mon Sep 15 12:24:10 2014
Thread-2: Mon Sep 15 12:24:12 2014
Thread-2: Mon Sep 15 12:24:14 2014
Exiting Thread-2
```

3.1.3 Senkronizasyon

`Threading` modülünde kolay uygulanabilen ve senkronizasyonu sağlayan bir kilit mekanizması vardır. Yeni bir kilit, `Lock()` metoduyla oluşturulur.

Yeni `lock` nesnesinin `acquire(blocking)` metodu iş parçacıklarının senkron olarak çalışmasını sağlar. Seçilimli olan `blocking` parametresi ise parçacığın kilidi çekmek için bekleyip beklemeyeceğini kontrol eder.

`blocking` 0 olduğu durumlarda, eğer kilit çekilemezse iş parçacığı hemen çalışacaktır. 1 durumunda ise parçacık bloklanmış modda olup, kilidin açılmasını bekleyecektir.

`release()` metodu kilide artık ihtiyaç duyulmadığında kilidi bırakmak için kullanılır.

threading_example.py

```
#!/usr/bin/env python3

import threading
import time

class myThread (threading.Thread):
    def __init__(self, threadID, name, counter, tLock):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
        self.tLock = tLock
    def run(self):
        print ("Starting", self.name)
        # Get lock to synchronize threads
        self.tLock.acquire()
        print_time(self.name, self.counter, 2)
        # Free lock to release next thread
        self.tLock.release()

def print_time(threadName, delay, counter):
    while counter:
        time.sleep(delay)
        print ("%s: %s" % (threadName, time.ctime(time.time())))
        counter -= 1

threadLock = threading.Lock()
threads = []

# Create new threads
thread1 = myThread(1, "Thread-1", 2, threadLock)
thread2 = myThread(2, "Thread-2", 1, threadLock)

# Start new Threads
thread1.start()
thread2.start()

# Add threads to thread list
threads.append(thread1)
threads.append(thread2)

```

```
# Wait for all threads to complete
for t in threads:
    t.join()
print("Exiting Main Thread")
```

çıktı

```
Starting Thread-1
Starting Thread-2
Thread-2: Mon Sep 15 14:58:01 2014
Thread-2: Mon Sep 15 14:58:03 2014
Thread-2: Mon Sep 15 14:58:05 2014
Thread-1: Mon Sep 15 14:58:06 2014
Thread-1: Mon Sep 15 14:58:07 2014
Thread-1: Mon Sep 15 14:58:08 2014
Exiting Main Thread
```

3.1.4 Mesajlaşma

`Queue` modülü yeni bir sıra nesnesini oluşturmaya ve belirli sayıda eleman tutmasını sağlar. Bu modülü kontrol eden metodlar aşağıdaki gibidir:

- `get()`: Sıradan bir elemanı alır ve döndürür.
- `put()`: Sıraya bir eleman koyar.
- `qsize()` : Sıra içindeki elemanların sayısını döndürür.
- `empty()`: Sıra boşsa `True` döndürür.
- `full()`: Sıra tamamen doluysa `True` döndürür.

threading_example.py

```
#!/usr/bin/env python

import queue
import threading
import time

exitFlag = 0

class myThread (threading.Thread):
    def __init__(self, threadID, name, q):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.q = q
    def run(self):
        print("Starting " + self.name)
        process_data(self.name, self.q)
        print("Exiting " + self.name)

def process_data(threadName, q):
    while True:
```

```
        queueLock.acquire()
        if not q.empty():
            data = q.get()
            queueLock.release()
            if data == "Quit":
                print("%s received quit request" % (threadName))
                break
            print("%s processing %s" % (threadName, data))
        else:
            queueLock.release()
        time.sleep(1)

threadList = ["Thread-1", "Thread-2", "Thread-3"]
nameList = ["One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight"]
queueLock = threading.Lock()
workQueue = queue.Queue(10)
threads = []
threadID = 1

# Create new threads
for tName in threadList:
    thread = myThread(threadID, tName, workQueue)
    thread.start()
    threads.append(thread)
    threadID += 1

# Fill the queue
queueLock.acquire()
for word in nameList:
    workQueue.put(word)
queueLock.release()

# Wait for queue to empty
while not workQueue.empty():
    pass

# Notify threads it's time to exit
for tName in threadList:
    workQueue.put("Quit")

# Wait for all threads to complete
for t in threads:
    t.join()
print("Exiting Main Thread")
```

çıktı

```
Starting Thread-1
Starting Thread-2
Starting Thread-3
Thread-3 processing One
Thread-1 processing Two
Thread-2 processing Three
Thread-3 processing Four
Thread-1 processing Five
Thread-2 processing Six
Thread-3 processing Seven
Thread-1 processing Eight
Thread-2 received quit signal
Exiting Thread-2
Thread-3 received quit signal
Exiting Thread-3
```

```
Thread-1 received quit signal
Exiting Thread-1
Exiting Main Thread
```

3.2 Forks - `multiprocessing` modülü

`multiprocessing` modülü paralel programlama için gerekli temel sınıflarla beraber gelir. Bunlardan en önemli üç tanesi `Process`, `Queue` ve `Lock` sınıflarıdır.

3.2.1 Process

`Process` bir pythonsüreci oluşturan bir soyutlamadır. Süreci oluşturmak ve çalışmasını kontrol etmek için gerekli metodları içerir. Örnek bir kullanım şöyledir:

```
from multiprocessing import Process

def say_hello(name='world'):
    print "Hello, %s" % name

p = Process(target=say_hello)
p.start()
p.join()
```

`Process` sınıfı çağrılır, içinde koşacak fonksiyon yazılır ve bu fonksiyonla bir nesne oluşturulur. `p.start()` metodu çağrılmadan hiçbir şey olmayacaktır. Son olarak da `p.join()` metodu çağrılarak sürecin sonlandırılması gerektiğini söyleriz. Bu metod olmadan alt süreç (child process) idle durumuna düşüp sonlanmayacak, zombi olacaktır.

Sürece parametre göndermek için `args` kelimesi kullanılır:

```
from multiprocessing import Process

def say_hello(name='world'):
    print "Hello, %s" % name

p = Process(target=say_hello, args=('Daniel',))
p.start()
p.join()
```

İstenildiği kadar alt süreç oluşturulup sonlandırılabilir. Ancak unutulmamalıdır ki, bilgisayarın performansını yükseltecek bir sınır vardır. Aşırı sayıda alt süreç oluşturma “fork bomb” denilen virüse neden olabilir. İşletim sisteminin buna karşılık bir koruması yoksa, DoS atağına maruz kalmış olur.

3.2.2 Queue

Bu nesneler process/thread korumalı FIFO yapılarıdır. İçinde çağrılabilir her türlü pythonnesnelerini saklayabilirler. Süreçler arasında veri paylaşımı için kullanılırlar.

```
from multiprocessing import Queue

q = Queue()

q.put('Why hello there!')
q.put(['a', 1, {'b': 'c'}])

q.get() # Returns 'Why hello there!'
q.get() # Returns ['a', 1, {'b': 'c'}]
```

3.2.3 Lock

`Queue` nesneleri gibi kullanılırlar. Kod parçasının kilidi alarak, süreç bitene ya da kilit açana kadar diğer süreçlerin benzer kodu çalıştırmalarını engeller.

```
from multiprocessing import Lock

l = Lock()

l.acquire()
print 'Ha! Only I can write to stdout!'
l.release()
```

3.2.4 Örnek uygulama

Aşağıdaki uygulama belirli web sayfalarının durumlarını paralel olarak sorgulayabilen bir pythonprogramıdır.

```
#!/usr/bin/env python
import httplib2
from multiprocessing import Lock, Process, Queue, current_process

def worker(work_queue, done_queue):
    try:
        for url in iter(work_queue.get, 'STOP'):
            status_code = print_site_status(url)
            done_queue.put("%s - %s got %s." % (current_process().name, url, status_code))
    except Exception, e:
        done_queue.put("%s failed on %s with: %s" % (current_process().name, url, e.message))
    return True


def print_site_status(url):
    http = httplib2.Http(timeout=10)
    headers, content = http.request(url)
    return headers.get('status', 'no response')
```

```
def main():
    sites = (
        'http://penny-arcade.com/',
        'http://reallifecomics.com/',
        'http://sinfest.net/',
        'http://userfriendly.org/',
        'http://savagechickens.com/',
        'http://xkcd.com/',
        'http://duelinganalogs.com/',
        'http://cad-comic.com/',
        'http://samandfuzzy.com/',
    )
    workers = 2
    work_queue = Queue()
    done_queue = Queue()
    processes = []

    for url in sites:
        work_queue.put(url)

    for w in xrange(workers):
        p = Process(target=worker, args=(work_queue, done_queue))
        p.start()
        processes.append(p)
        work_queue.put('STOP')

    for p in processes:
        p.join()

    done_queue.put('STOP')

    for status in iter(done_queue.get, 'STOP'):
        print status

if __name__ == '__main__':
    main()
```

Bu kodu tek süreç halinde yazabilirdik. Hatta `print_site_status` fonksiyonu aslında tam olarak ne yapılması gerektiğini söylüyor.

`main()` fonksiyonumuzda kontrol etmek istediğimiz site isimleri yazmaktadır. Bu URL adreslerini bir `Queue` içine atıyoruz. Böylelikle alt süreçlerin de bu adreslere ulaşabileceği ortak bir kaynağı oluyor. Bundan sonra belirtilmiş sayı kadar olan alt süreçlerimizi başlatıyoruz. Bunları daha sonra kontrol edebilmek için bir listede tutuyoruz. Bütün alt süreçler başladıktan sonra her süreç sonrasına bir "STOP" belirteci koyuyoruz ve süreçleri `join` ile ana sürece bağlıyoruz. Bütün süreçler tamamlandıktan sonra başka bir `Queue` olan `done_queue` içinden sitelerin durumlarını okuyoruz.

3.3 Ödev IV - Çok parçalı Caesar Cipher

Bu ödevde sizden en basit şifreleme metodu olan tek alfabeli (monoalphabetic) bir şifreleme algoritması yazmanız beklenmektedir. Toplamda iki ayrı versiyon yazılıp, birinde thread, diğerinde fork yapısı kullanılacaktır.

3.3.1 Algoritma tanımı (Caesar Cipher)

Algoritmanız, kolaylaştırmak için sadece İngiliz alfabesini kullanacak, verilen metinde büyük-küçük harf ayrımı yapılmayacak, girdi metninin (plain text) küçük harlerden oluştuğu farzedilip işlenecek karakterler küçük harfe dönüştürülecektir (`"".lower()`). Alfabeyi oluşturan 26 harf dışındaki karakterler (boşluk, noktalama işaretleri, satır sonu karakteri gibi) şifrelemeye dahil edilmeyip aynen bırakılacaktır (şifrelenmiş metinde yer alacaklar).

Caesar Cipher, kullanılan alfabenin kaydırılarak girdi metnindeki harflerin karşılık gelen harfe dönüştürülmesi şeklinde tanımlanır. Anahtar alfabe sıralı alfabenin belirli sayıda karakter sağa ya da sola kaydırılmasıyla oluşturulur. Biz dairesel sola kaydırma kullanacağız. Örnek bir anahtar aşağıdaki gibidir, alfabe 7 karakter sola kaydırılarak oluşturulmuştur:

Anahtar oluşturma ($s = 7$)

```
abcdefghijklmnopqrstuvwxyz
HIJKLMNOPQRSTUVWXYZABCDEFG
```

Yukarıdaki anahtara göre şifrelenen bir mesaj örneği aşağıda verilmiştir:

Şifreleme geçişi

```
Plain text....: lorem ipsum dolor sit amet
Crypted text..: SVYLT PWZBT KVSVY ZPA HTLA
```

Parametreler: Yazılacak program anahtar oluşturulması için alfabenin kaç basamak sola kaydırılacağını (`s`) parametre olarak alacak ve anahtarı oluşturacak. Program çok iş parçalı olacağı için, kaç parçalı olacağı da programa parametre (`n`) olarak verilecektir. Her iş parçası (thread veya child process) belirli bir uzunlukta metin bloğunu okuyup şifreleyecektir. Bu uzunluk da parametre olarak verilecektir (`l`).

- `s`: Kaydırma (shift) parametresi. Anahtarın sola doğru ne kadar kaydırılacağını gösterir.
- `n`: Alt iş sayısı. Şifreleme işi için kaç tane alt iş açılacağını gösterir, (child process veya thread sayısı).
- `l`: Blok uzunluğu (length). Bir alt işte, bir kerede ne kadar uzunlukta şifreleme yapılacağını gösterir.

Girdi dosyası: Metin bir girdi dosyasından okunacak (`input.txt`) ve verilen anahtara göre şifrelenip başka bir dosyaya yazılacak. Girdi dosyası, programlarınızla aynı dizinde olmak üzere adı programın içinde gömülü olarak yazıyor olacak, ayrıca komut parametresi olarak verilmeyecek.

Girdi metni şuradan edinilebilir: The Raven by Edgar Allen Poe

3.3.2 Örnek senaryo

Örnek metin

```
lorem ipsum dolor sit amet
```

Girdi metninin yukarıdaki gibi olduğunu farzedelim. Parametrelerimiz de `s`=3, `n`=4 ve `l`=5 şeklinde olsun.

Bu senaryoya göre anahtarımız "DEFGHIJKLMNOPQRSTUVWXYZABC" olacaktır. Thread programlaması yapıyorsak toplamda 4 tane thread açacağız ve her thread'de bir kerede 5 karakter şifreleyeceğiz. t_i, thread adını gösterecek şekilde, ilk aşamada (sıralaması farklı şekilde olması çok muhtemel) t_1 "lorem", t_2 " ipsu", t_3 "m dol" ve t_4 "or si" karakter dizilerini şifreleyecektir. Daha sonra ilk tamamlanan thread bir sonraki 5 karakteri ("t amet") alıp devam edecektir. Dikkat: Threadler aynı anda çalışacağı için çalışma sıralaması önemli değildir. Ama şifrelenmiş karakter dizisini çıktı dosyasına sıralı şekilde yazdırmak gerekmektedir. Bunun için threadler arasında senkronizasyon sağlanmalıdır. Sırası gelmeyen bloğu şifrelemiş bir thread dosyaya yazabilmek için yazılacak bloğun sırasının gelmesini beklemelidir. Bunun için kuyrukta şifrelenen bloğun kaçıncı sırada olduğunu da tutmak faydalı olabilir.

Önemli: Dosyadan okuma ve dosyadan yazma işlemleri ana program (main thread) tarafından yapılacak.

Önemli: Her karakter bloğu için yeni thread açılmayacak, açılan thread bitene kadar döngüde çalışacaktır. Yani, parametrelerde 4 tane thread açılması isteniyorsa, sadece 4 tane iş yapan thread açılacak. Şifrelenecek yeni bir şey kalmayıp, şifrelenmiş her şey dosyaya yazıldığında thread'ler bitirilecektir.

Önemli: Ödevleri kontrol ederken aşağıdaki şekilde bir parametrizasyona uymasını bekliyoruz. `sys.argv` kullanılacak:

Program çalıştırırken alacağı parametreler

```
$ python caesar_cipher_thread.py 5 7 9
```

3.3.3 Teslim edilecekler

1. Son teslim zamanı: **07.11.2020**- 22:00 (TSİ)
2. Github deponuzda `odev04` isimli yeni bir dizin altında 5 tane dosya olacak:
   (a) Girdi metnini içeren bir txt biçimli dosya: "`input.txt`"
   (b) Thread ile yazılmış programınızı içeren "`caesar_cipher_thread.py`" dosyası.
   (c) Fork ile yazılmış programınızı içeren "`caesar_cipher_fork.py`" dosyası.

(d) Verilen parametrelerle şifrelenmiş çıktıyı içeren txt biçimli dosyalar "`crypted_thread_<s>_<n>_<l>.txt`" ve "`crypted_fork_<s>_<n>_<l>.txt`" dosyası. `<s>`,`<n>` ve `<l>` yerinde sayısal parametreler yazıyor olacak. Bu dosyaların boş halleri asistan hocanız tarafından ilgili dizine önceden yerleştirilecektir. Hangi parametreleri kullanacağınızı o dosyanın isminden öğrenebilirsiniz.

Bölüm 4

Soket programlama - 07.11.2020

pythonağ servislerine iki katman halinde erişim sağlamaktadır. Alt katmanda, işletim sisteminin temel soket desteğine erişilebilir. Bu hem bağlantı temelli (connection-oriented) hem de bağlantısız (connectionless) protokoller için istemci ve sunucu yazmayı sağlar.

Üst katman için ise, python'un FTP ve HTTP gibi uygulama seviyesi ağ protokollerine yüksek seviye erişim hizmeti sağlar.

4.1 Soketler

Soketler çift taraflı iletişim bağlantısının uç noktalarıdır. Bir süreç içinde, aynı makinadaki süreçler arasında veya bir şekilde birbirleriyle iletişim kurabilen uzak makinaların süreçleri arasında iletişirler.

Farklı kanal tipleri üzerinde çalışabilirler: Unix domain sockets, TCP, UDP gibi. `socket` kütüphanesi genel bir arabirim ile birlikte yaygın iletişimi sağlayacak özel sınıfları içerir.

Bu kütüphanenin kendi kelime haznesi vardır:

- `domain`: İletim mekanizması olarak kullanılacak protokoller ailesi. Bu sabitler: AF_INET, PF_INET, PF_UNIX, PF_X25, vs.
- `type`: İki uç nokta arasındaki bağlantı tipini belirler. Bağlantı temelli protokol için genellikle SOCK_STREAM, bağlantısız protokoller için SOCK_DGRAM kullanılır.
- `protocol`: Genellikle sıfır. `domain` ve `type` içinde protokol çeşidini tanımlamak için kullanılabilir.

- `hostname`: Ağ arabiriminin tanımlayıcısı.
  - Bilgisayar adı, ağ IP adresi veya IPv6 adresi içeren bir karakter katarı
  - Yayın (broadcast) belirleyen bir INADDR_BROADCAST adresi içeren karakter katarı
  - INADDR_ANY betimleyen sıfır boyutlu bir karakter katarı
  - İkilik bir adres tanımlayan bir tam sayı.
- `port`: Her sunucu istemcilerini bir veya birden fazla porttan dinler. Port numarası tanımlayan bir tamsayı, karakter katarı veya servis adından oluşabilir.

4.2 `socket` modülü

Yeni bir soket oluşturmak için `socket.socket()` fonksiyonu kullanılır.

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

- `socket_family`: AF_UNIX veya AF_INET.
- `socket_type`: SOCK_STREAM veya SOCK_DGRAM.
- `protocol`: Genellikle kullanılmaz, varsayılan 0'dır.

`socket` nesnesi oluşturulduğunda gerekli fonksiyonlarına ulaşabilir hale geliriz.

4.2.1 Sunucu metodları

- `s.bind()`: Verilen adresi (bilgisayar adı ve port numarası çiftini) sokete bağlar.
- `s.listen()`: TCP dinleyicisini kurup başlatır.
- `s.accept()`: Pasif TCP istemci bağlantısını kabul eder, bağlantı gelene kadar bekler (bloking).

4.2.2 İstemci metodları

- `s.connect()`: Aktif olarak TCP bağlantısını başlatır.

Genel Metodlar

- `s.recv()`: TCP mesajını alır.
- `s.send()`: TCP mesajını gönderir.
- `s.recvfrom()`: UDP mesajını alır.
- `s.sendto()`: UDP mesajını iletir.
- `s.close()`: Soketi kapatır.
- `s.gethostname()`: Bilgisayar adını döndürür.

4.3 Basit sunucu örneği

simple_server.py

```
1  #!/usr/bin/env python
2
3  # Import socket module
4  import socket
5
6  # Create a socket object
7  server_socket = socket.socket()
8
9  # Get local machine name
10 host = "0.0.0.0"
11
12 # Reserve a port for your service.
13 port = 12345
14
15 # Bind to the port
16 server_socket.bind((host, port))
17
18 # Enable the server
19 server_socket.listen(5)
20
21 while True:
22     # Wait and establish connection with client.
23     conn_socket, addr = server_socket.accept()
24
25     print('Got connection from', addr)
26     conn_socket.send(b'Thank you for connecting!\n')
27     # Close the connection
28     conn_socket.close()
```

istemci olarak telnet

```
$ telnet localhost 12345
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Thank you for connecting!Connection closed by foreign host.
```

sunucu çıktısı

```
$ ./simple_server.py
Got connection from ('127.0.0.1', 46005)
```

4.4 Basit istemci örneği

simple_client.py

```
1  #!/usr/bin/env python
2
3  # Import socket module
4  import socket
5
6  # Create a socket object
7  s = socket.socket()
8
9  #
10 host = <baglanilacak adres>
11 port = 12345
12
13 s.connect((host, port))
14 print(s.recv(1024).decode())
15 s.close()                    # Close the socket when done
```

istemci çıktısı

```
$ ./simple_client.py
Thank you for connecting
```

4.5 Ödev - Basit Mesajlaşma

Uygulama zamanı saatinde sizden beklenen aralarında mesajlaşan bir sunucu ve bir istemci düzenlemek.

- Sunucu:
  - Ana thread her zaman bağlantı bekleme modunda çalışacak.
  - Yeni bir bağlantı geldiği anda bir thread açarak istemciyle o thread üzerinden haberleşecek. Böylelikle birden fazla istemciye threadler açarak cevap verebilecek.
  - Kendisine bağlanan istemciye bağlandığı anda saati söyleyecek."`Saat şu an <saat>`"
  - Sunucu, kendisine söylenen 5 tane kelimeyi tanıyor olacak:
    1. `Selam`
    2. `Naber`
    3. `Hava`

4. `Haber`
5. `Kapan`

– Yukarıdakilerin karşılığı olarak aşağıdakileri söyleyecek.

1. `Selam`
2. `Iyiyim, sagol`
3. `Yagmurlu`
4. `Korona`
5. `Gule gule`

– 5. lafını söyledikten sonra ise istemciyle olan bağlantısını kesecek.

– Tanımadığı bir kelime geldiğinde `Anlamadim` diyecek.

– Bağlantıyı kestiğinde ilgili thread de sonlandırılacak.

- İstemci:

– Bağlantı adresini ve portunu komut satırından alacak.

– Program başlarken bağlantı kurulacak.

– Tek thread halinde çalışacak.

– Kullanıcıdan giriş bekleyip, gelen girişleri sunucuya gönderecek.

– Sunucuya bir şey gönderdikten sonra cevap gelmesini bekleyecek ve gelen cevabı ekrana basacak.

Uyarı: Verilen iskelet kodlar hatalı olabilir, fikir vermesi amacıyla sizlere verilmiştir.

4.5.1 Teslim edilecekler

1. Son teslim zamanı: **11.11.2020**- 23:59 (TSİ)
2. Gitlab sayfanızdaki `dagitik` deponuzda `odev05` isimli bir dizin altında 4 tane dosya olacak:
   (a) Sunucu programınızı içeren "`odev05_sunucu.py`" dosyası.
   (b) İstemci programınızı içeren "`odev05_istemci.py`" dosyası.
   (c) Örnek çıktıların yer aldığı iki pdf biçimli dosya: "`cikti_sunucu.pdf`" ve "`cikti_istemci.pdf`".

Bölüm 5

IRC tipi Mesajlaşma - 25.11.2020

5.1 Kısa tanım

Bu çalışmada sizden beklenen günümüzde eskimiş bir teknoloji olarak kabul edilen ama hala kullanımda olan IRC (Internet Relay Chat) BENZERI bir mesajlaşma sunucusu ve istemcisi yazmak. Yazacağınız yazılım ikilisi (sistem) aşağıdaki fonksiyonları içermelidir:

- Sunucu ve istemci arasında alt seviye haberleşmeyi sağlamak için bir uygulama protokolü tasarlamak gerekiyor.
- Sistem tek chat odasından oluşacak. Bağlanan kullanıcılar istediklerinde kimlerin bağlanmış olduğunu görebilecekler.
- Gönderilen genel mesajlar bütün bağlı kullanıcılara gönderilecek.
- Kullanıcılar kişiye özel mesaj gönderebilecek. Bu mesaj sadece hedefteki kullanıcıya iletilecek.
- İstemci arayüzünde mesaj gönderme ve mesajları görmek için iki ayrı bölge olacak.

5.2 Uygulama Protokolü

Sunucu ve istemci arasında alt seviye haberleşmeyi sağlamak için bir uygulama protokolü kullanılması gereklidir. Normalde bu standartlar RFC denilen dokümanlarda anlatılmaktadır. Burada bütün mesajlar RFC standartlarından uzak sade bir dille anlatılıyor. Uygulama protokolümüzde iki bağlantı yönü, dolayısıyla iki farklı protokol yapısına ihtiyacımız var. Biri istemciden sunucuya,

diğeri sunucudan istemciye giden mesajlardır. Bunlar aynı görünseler de biraz farklı çalışmaktadırlar.

5.2.1 İstemciden sunucuya mesaj gönderme

İstemci-sunucu yönlü mesajlaşma soru cevap şeklinde olacaktır ve her mesaj satır sonu karakteri (\n) ile bitirilecektir. Komut kökleri üç karakterden oluşacaktır.

Yeni bağlantı:

Bağlantı kurduktan sonra sunucuya kullanıcı ismi aktarılacaktır. Kullanıcı ismi boşluksuz Türkçe karakter içermeyen bir yapıda olmalıdır. Sunucunun kullanıcı ismini beğenmediği durumlarda veya kullanıcı ismi mevcutsa sunucu girişi reddedip bağlantıyı kapatmalıdır.

Sorgu: istemci→sunucu

```
NIC <rumuz>
```

Kullanıcıyı kabul et:

Cevap: sunucu→istemci

```
WEL <rumuz>
```

Böyle bir kullanıcı mevcutsa reddet:

Cevap: sunucu→istemci

```
REJ <rumuz>
```

Bağlantıyı kapat:

Sorgu: istemci→sunucu

```
QUI
```

Cevap: sunucu→istemci

```
BYE <rumuz>
```

Kullanıcı listesi sorgulama

Sorgu: istemci→sunucu

```
GLS
```

Cevap: sunucu→istemci

```
LST <noktali virguller ayrilmis rumuzlar>
```

Örnek Cevap: sunucu→istemci

```
LST crazy_boy82:stayla90
```

Bağlantı Kontrol

Sorgu: istemci→sunucu

```
PIN
```

Cevap: sunucu→istemci

```
PON
```

Genel mesaj

Sorgu: istemci→sunucu

```
GNL <mesaj>
```

Cevap: sunucu→istemci

```
OKG
```

Özel mesaj

Sorgu: istemci→sunucu

```
PRV <nickname>:<mesaj>
```

Cevap: sunucu→istemci

```
OKP
```

Gönderilecek kullanıcı mevcut değilse:

Cevap: sunucu→istemci

```
NOP <rumuz>
```

Hata durumları

İstemciden sunucuya giriş yapmadan, giriş yapma isteği (NIC), bağlantı kontrolü isteği (PIN) ve çıkış isteği (QUI) dışında bir sorgu iletilirse sunucu cevap olarak aşağıdaki mesajı gönderecektir:

Cevap: sunucu→istemci

```
LRR
```

İstemciden sunucuya giden ve protokole uygun olmayan mesajlarda istemci aşağıdaki mesajı dönecektir:

Cevap: sunucu→istemci

```
ERR
```

Tablo 5.1: İstemci sorguları için protokol özeti

İstek	Parametre	Cevap	Parametre	Tanım
NIC	rumuz	WEL	rumuz	Yeni kullanıcı kabulü
		REJ	rumuz	Yeni kullanıcı reddi
QUI		BYE	rumuz	Çıkış
GLS		LST	rumuz:rumuz:...	Kullanıcı listesi isteme
PIN		PON		Bağlantı testi
GNL	mesaj	OKG		Genel mesaj gönderme
PRV	rumuz:mesaj	OKP		Özel mesaj gönderme
		NOP	rumuz	Kullanıcı bulunamadı
Komut		ERR		Hatalı komut
Komut		LRR		Giriş yapılmadı

5.2.2 İstemci tarafında örnek bir konuşma

Kalın harflerle yazılanlar istemciden sunucuya olan sorguları, normal harflerle yazılanlar sunucudan istemciye gelen cevapları belirtilmektedir.

örnek protokol konuşması

```
NIC serhan
REJ serhan
NIC ceren
WEL ceren
GLS
LST ahmet:ali:ceren:serhan
HEB
ERR
PRV ali:Merhaba ali
OKP
PIN
PON
GNL Merhaba Dunya
```

```
OKG
QUI
BYE ceren
```

5.2.3 Sunucudan istemciye mesaj gönderme

İstemci→sunucu yönlü mesajlaşma soru cevap şeklinde olacaktır ve her mesaj satır sonu karakteri (\n) ile bitirilecektir. Komut kökleri üç karakterden oluşacaktır. Mesaj detayları aşağıda belirtilmektedir.

Genel mesaj

sunucu sorgusu

```
GNL <rumuz>:<mesaj>
```

istemci cevabı

```
OKG
```

Özel mesaj

sunucu sorgusu

```
PRV <rumuz>:<mesaj>
```

istemci cevabı

```
OKP
```

Sistem mesajı

sunucu sorgusu

```
WRN <mesaj>
```

istemci cevabı

```
OKW
```

Bağlantı Kontrol

Sorgu: istemci→sunucu

```
TIN
```

Cevap: sunucu→istemci

```
TON
```

Tablo 5.2: Sunucu sorguları için protokol özeti

İstek	Parametre	Cevap	Parametre	Tanım
GNL	mesaj	OKG		Genel mesaj iletme
PRV	rumuz:mesaj	OKP		Özel mesaj iletme
WRN		OKW		Sistem mesajı iletme
TIN		TON		Bağlantı testi
Komut		ERR		Hatalı komut

5.3 İstemci - 25.11.2020

5.3.1 Komut işleyicileri

Gelen işleyici

İstemci tarafında iki farklı komut işleyici bulunmaktadır. Bunlardan ilki soket üzerinden gelen mesajları değerlendirip bunlara göre hareketler tanımlayan `incoming_parser`'dır. Bu işleyici Tablo 5.1 ve Tablo 5.2'da verilen protokol tanımlarını kullanarak sunucu tarafından üretilen mesajlara göre çalışır. Sunucu tarafından gönderilen istekleri ve istemcinin yaptığı isteklerin cevaplarını karşılayarak istemci ekranına yansıması için hazırlar.

Gelen mesajları işleyeceğinden `ReadThread` içinde bir metod olarak tanımlanması tavsiye edilir.

Giden işleyici

İstemci arayüzünde basit bir komut satırı bulunmaktadır. Bu komut satırı mesaj veya komut yazmaya yarar. Aşağıda yazılabilecek komutlar tanımlanmıştır:

- `/user <nickname>`: Yeni kullanıcı tanımlama.
- `/list`: Kanal listesi isteme.

- `/quit`: Kanalı terketme.
- `/msg <nickname> <message>`: Özel mesaj gönderme.

Başına "/" koyulmayan her şey genel mesaj olarak algılanacaktır. Komut satırından okunan veriler giden işleyiciden (`outgoing_parser`) geçirilip sokete yazılacak protokol verisi üretilir ve ilgili kuyruğa yazılır.

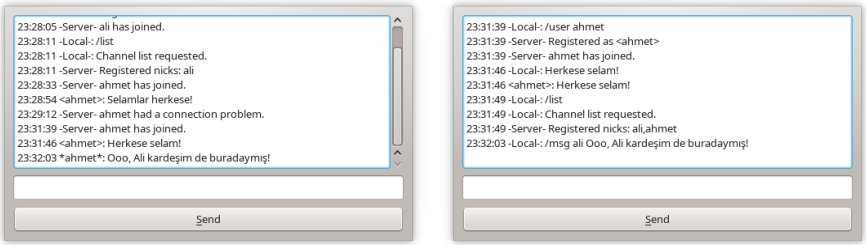
5.3.2 Threadler

İstemciniz mesajların ASENKRON gönderilebilmesi için iki thread çalıştıracaktır: `ReadThread` ve `WriteThread`. İlki soket üzerinden gelen mesajları yakalayıp işleyecek ve bunlardan anlamlı bilgiler çıkarıp kuyruk yapısına yazacak, ikincisi ise gönderilmek istenen verilerin ilgili kuyruk yapısından okunarak sokete yazılmasını sağlayacaktır.

Bu iki thread yanında görsel arayüzü kontrol eden Qt tabanlı başka bir thread bulunmaktadır.

5.3.3 İstemci arayüzü

İstemci arayüzü kodları Qt API kullanılarak yazılacaktır. İlgili kaynaklar asistan hocanız tarafından sağlanacaktır.



Şekil 5.1: İstemci örnek görünümü

client

```
import sys
import socket
import threading
from PyQt5.QtCore import *
from PyQt5.QtGui import *
import queue
import time

class ReadThread (threading.Thread):
    def __init__(self, name, csoc, threadQueue, app):
        threading.Thread.__init__(self)
        self.name = name
```

```
        self.csoc = csoc
        self.nickname = ""
        self.threadQueue = threadQueue
        self.app = app

    def incoming_parser(self, data):
        ...

    def run(self):
        ...
    ...


class WriteThread (threading.Thread):
    def __init__(self, name, csoc, threadQueue):
        threading.Thread.__init__(self)
        self.name = name
        self.csoc = csoc
        self.threadQueue = threadQueue

    def run(self):
        ...

class ClientDialog(QDialog):

    ''' An example application for PyQt. Instantiate
        and call the run method to run. '''
    def __init__(self, threadQueue):

        self.threadQueue = threadQueue

        # create a Qt application --- every PyQt app needs one
        self.qt_app = QApplication(sys.argv)

        # Call the parent constructor on the current object
        QDialog.__init__(self, None)

        # Set up the window
        self.setWindowTitle('IRC Client')
        self.setMinimumSize(500, 200)

        # Add a vertical layout
        self.vbox = QVBoxLayout()

        # The sender textbox
        self.sender = QLineEdit("", self)

        # The channel region
        self.channel = QTextBrowser()

        # The send button
        self.send_button = QPushButton('&Send')

        # Connect the Go button to its callback
        self.send_button.clicked.connect(self.outgoing_parser)

        # Add the controls to the vertical layout
        self.vbox.addWidget(self.channel)
        self.vbox.addWidget(self.sender)
        self.vbox.addWidget(self.send_button)

        # start timer
```

```
        self.timer = QTimer()
        self.timer.timeout.connect(self.updateText)
        # update every 100 ms
        self.timer.start(10)

        # Use the vertical layout for the current window
        self.setLayout(self.vbox)


    def updateText(self):
        if not self.screenQueue.empty():
            data = self.screenQueue.get()
            t = time.localtime()
            pt = "%02d:%02d:%02d" % (t.tm_hour, t.tm_min, t.tm_sec)
            self.channel.append(pt + " " + data)
        else:
            return

    def cprint(self, data):
        ...
        self.channel.append(data)

    def outgoing_parser(self):
        ...

    def run(self):
        ''' Run the app and show the main form. '''
        self.show()
        self.qt_app.exec_()


# connect to the server
s = socket.socket()
host = ...
port = ...
s.connect((host,port))

sendQueue = ...

app = ClientDialog(sendQueue)

# start threads
rt = ReadThread("ReadThread", s, sendQueue, app)
rt.start()

wt = WriteThread("WriteThread", s, sendQueue)
wt.start()

app.run()

rt.join()
wt.join()
s.close()
```