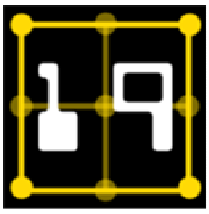
T.C. ONDOKUZ MAYIS ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ



PYTHON İLE BİLGİSAYAR PROGRAMLAMAYA GİRİŞ

HAZIRLAYAN

Arş. Gör. Emre Gürbüz

PYTHON PROGRAMLAMA DİLİ REHBERİ

LABORATUVAR ÇALIŞMASI – 1

Bu Çalışmanın Amacı

Bu çalışmadaki amacımız, kullandığımız sistem ve kullanım hedefimiz için en uygun olan Python sürümünü bilgisayarımıza yükleyerek bilgisayarımızı programlamaya başlamak için hazır hale getirmektir.

Kurulum yaptıktan sonraki amacımız, kurulumunu yapmış olduğumuz, Python yazılım geliştirme aracı olan IDLE ortamında örnek programlar hazırlayarak Python programlama diline giriş yapmaktır.

Hangi Python Bizim İçin Daha Uygun?

Pek çok popüler Linux sürümünde, ayrıca Macintosh OS X 10.2 ve sonrasındaki sürümlerinde işletim sistemiyle birlikte gelmesine rağmen Windows ile beraber gelmeyen Python' u, hemen hemen her işletim sistemi (Windows, Macintosh OS, tüm Linux sürümleri, Sun Solaris vs.) ile beraber kullanmak mümkündür. Hangi Python' un bizim için daha uygun olduğu sorusunun cevabı kullandığımız işletim sisteminde yatmaktadır.

Windows' ta Python

Ücretsiz olarak temin edilebilen ve açık kaynak kodlu olan Python' un en güncel halini indirerek bilgisayarımıza kurmak için aşağıdaki adımları takip etmeliyiz:

1. Python Windows yükleyicisini indirmek için, http://www.python.org/download/releases/ adresine girelim.
2. Python 2.6.4' e tıklayalım.
3. Çıkan sayfanın altında, Windows için kurulum dosyasını indirelim (32 Bit' lik mimariye sahip bilgisayarlar için "Windows x86 MSI Installer", 64 Bit' lik mimariye sahip olanlar içinse "Windows AMD 64 MSI Installer" indirilecektir).

4. “msi” uzantılı kurulum programını çalıştıralım, “ileri” diyerek kurulumu tamamlayalım.
5. ‘**Başlat → Programlar**’ altında “Python” u seçip, “IDLE (Python GUI)” a tıklayalım. “Python Shell” penceresi açılacaktır. Kurulum tamamlanmıştır.

Linux’ ta Python (Ubuntu 9.10)

İşletim sisteminin kurulumu ile ilk başta bilgisayarımıza yüklenmiş olmamasına rağmen program ekleme özelliğini kullanarak Ubuntu’ yu yükleyebiliriz. Bunun için aşağıdaki işlemleri sırayla takip edelim:

1. ‘**Uygulamalar → Ubuntu Yazılım Merkezi**’ seçimini yapalım.
2. Sağ üstte çıkan arama alanına “idle” yazalım.
3. “IDLE (Python-2.6)” yı seçerek sağ tarafında çıkan oka basalım. Gelen ekrandaki “Kur” düğmesine basarak kurulmasını sağlayalım.
4. Kurulum tamamlandıktan sonra, “Uygulamalar” altında “Programlama” bölümü oluşacaktır. ‘**Uygulamalar → Programlama → IDLE**’ seçeneğine tıklayalım. “Python Shell” penceresi açılacaktır. Kurulum tamamlanmıştır.

Python’ a Giriş

Python programları bir yorumlayıcı (interpreter) tarafından çalıştırıldıkları için Python, “yorumlanan programlama dilleri” kapsamına girer. Python yorumlayıcısını kullanmanın iki farklı yolu vardır: **etkileşimli olarak** (interactive mode) ve **betik kullanarak** (script mode). [betik: yazılı olan şey]

Etkileşimli kullanımda, biz kodumuzu yazdıktan sonra yorumlayıcı bize sonucu döndürür:

```
>>> (6 * 2) - (21 / 3) + 4
9
```

Komut ekranındaki >>> işareti bize yorumlayıcının hazır durumda olduğunu bildirmektedir.

NOT: Bölen ve bölünenin birer tamsayı olduğu bir bölme işleminde kalan varsa bile sonuca etki etmez. Ancak kalanlı bir bölme işleminde bölen ve bölünenden en az biri ondalıklı bir

sayı olarak seçilirse Python yorumlayıcısı **ondalıklı bölme** yapacağı için sonuç da ondalıklı olacaktır. Aşağıdaki örneği incelemek bu konuyu daha da somutlaştıracaktır:

```
>>> 10 / 3
3
>>> 10.0 / 3
3.3333333333333335
>>> 10 / 3.0
3.3333333333333335
>>> 10.0 / 3.0
3.3333333333333335
```

Betik kullanarak program yazdığımızda ise yazılan program bir metin dosyasında saklanır. Bu dosyayı çalıştırdığımız zaman içerisinde bulunan tüm kod yorumlanır ve sonucu döndürülür. Kabul edilen bir gösterim biçimi olarak, Python programlarının yer aldığı betik dosyalarına "**py**" uzantısı verilir.

Bir betik dosyası oluşturarak içerisine bir Python programı yazmak için şu aşamaları takip edebiliriz:

1. **Python Shell** üzerinde '**File → New Window'** diyerek boş bir sayfa açarız.
2. Bu sayfaya, uygun sözdizimiyle kodlanmış bir Python kodu yazdıktan sonra, sayfa üzerinden '**File → Save**' diyerek dosyamıza bir isim verir ve kaydederiz (Uzantısının "py" olmasına dikkat etmeliyiz.).
3. Sayfamız açıkken üst taraftaki menüden '**Run → Run Module**' seçerek ya da **F5** tuşuna basarak dosyadaki kodun yorumlanmasını ve sonucun dönmesini sağlarız.
4. Bu pencere kapalıyken de Python Shell ekranından bu pencereyi tekrar açarak çalıştırabiliriz. Bunun içinse, '**File → Open**' diyerek kaydetmiş olduğumuz dosyayı gösterir ve bu dosyalara ait kodların bulunduğu ekranı yeniden açarız. Daha sonra, 3. maddeyi tekrar ederiz.

Python Hakkında Yardım Alma

Python ile programlama yaparken, bilemediğimiz ya da emin olamadığımız konularda bize yardımcı olabilecek olan kaynaklardan yararlanmamız; Python' un programlama dünyasına daha başarılı bir giriş yapmamız, dili daha doğru ve çabuk bir biçimde öğrenmemiz ve daha

başarılı programlar geliştirmemiz konularında bize büyük katkı sağlayacaktır. "Kaynaklar" bölümünde bize yardımcı olacak bu kaynaklara ulaşabiliriz.

Örneğin, [2]' den, kullanmakta olduğumuz Python sürümüne uygun olan Python kılavuzunu indirerek ihtiyacımız olduğunda faydalanabiliriz.

Ayrıca, Python Shell ekranında "**help()**" komutunu kullanarak da yardım almamız mümkündür. Bunun için, parantez içerisine, tırnak içinde yardım almak istediğimiz konuyu yazabiliriz. Örnek olarak, matematikle ilgili fonksiyonlar içeren "math" sınıfı ile ilgili yardım almak için komut satırına aşağıdaki ifade yazılabilir:

```
>>> help('math')
```

Değişken Tanımlama

"**Değişken**" i, bir niceliğin depolanabileceği bir yer, alan olarak tanımlayabiliriz. Değişkenler, içlerinde bir **karakter dizisi** (örneğin ismimiz) bulundurabileceği gibi bir **tamsayı** (örneğin yaşımız) ya da **ondalıklı sayı** (örneğin boyumuz) da bulundurabilirler. Değişkenlerin türleri de içlerinde bulundurdukları bu değerlere göre değişmektedir. Örneğin, Python yorumlayıcısını etkileşimli biçimde kullanarak, ismi "Emre", yaşı 23 ve boyu 1.83 olan bir kişinin bu bilgilerini saklamak üzere üç ayrı değişken tanımlayalım:

```
>>> isim = "Emre"
>>> yas = 23
>>> boy = 1.83
```

Şimdi bu değişkenlerin her birinde ilgili kişiye ait bazı değerler tutulmakta olup bunlardan birisi karakter dizisi şeklinde olan isim ("**isim**" değişkeni içerisinde), diğeri tamsayı şeklinde olan yaş ("**yas**" değişkeni içerisinde), bir diğeri ise ondalıklı sayı şeklinde olan boy ("**boy**" değişkeni içerisinde) bilgileridir.

Python' da bir değişkenin, ya da bir değişkene atanabilecek bir değerin türünü öğrenmek için "**type()**" fonksiyonu kullanılır. Türü öğrenilmek istenen değişken ya da değer, bu fonksiyonun parantezleri içerisine yazılır.

Yukarıdaki kutuda vermiş olduğumuz tanımlama örneği üzerinden (Python Shell penceresini kapatmadan) devam edecek olursak, "**type()**" fonksiyonunun kullanımı ile ilgili şu örneği verebiliriz:

```
>>> type(isim)
<type 'str'>
>>> type(yas)
<type 'int'>
>>> type(boy)
<type 'float'>
```

Örnekte görüldüğü gibi "**type()**" fonksiyonu bize değişkenlerin türlerini vermektedir. Burada **str** "karakter dizisi" ne, **int** "tamsayı" ya, **float** ise "ondalıklı sayı" ya karşılık gelmektedir.

type() fonksiyonu ile değişkenlerin türlerini öğrenebileceğimiz gibi, bu değişkenler içerisine atılacak olan değerlerin türlerini de öğrenmemiz mümkündür. Python Shell ekranını kapatmadan yukarıdaki örneğe devam ederek bunu görebiliriz:

```
>>> type("Emre")
<type 'str'>
>>> type(23)
<type 'int'>
>>> type(1.83)
<type 'float'>
```

Bütün bunları göz önünde bulundurursak "Python, pek çok programlama dilinden farklı olarak, **değişkenlerin türlerini, kendilerine atanan değerin türüne bakarak, atama işlemi sırasında dinamik olarak belirler**." diyebiliriz.

İçerisinde değer bulunan değişkenleri ise programda yer alan başka fonksiyonlarla birlikte kullanabiliriz. Örnek:

```
>>> universite = "Ondokuz Mayıs Üniversitesi"
>>> il_trafik_kodu = 55
>>> print universite
Ondokuz Mayıs Üniversitesi
>>> print il_trafik_kodu
55
```

Veri Girişi

Python' da bir değişkene değer atıp, daha sonra bu değişkeni, içerisine atılan değer yerine kullanmayı incelemiştik. Yani, "**universite = "Ondokuz Mayıs Üniversitesi"**" atamasını yaptıktan sonra, "**"Ondokuz Mayıs Üniversitesi"**" değeri yerine "**universite**" değişkenini kullanmamız bizi aynı sonuca götürüyordu.

Bu kısımda ise programın, çalışma esnasında kullanıcıdan bir veri alarak, aldığı bu veriyi işleme tabi tuttuktan sonra kullanıcıya geri dönüş yapmasını öğreneceğiz.

Python' da klavyeden veri almak için "**raw_input()**" fonksiyonu kullanılır. Bu fonksiyonda parantezler arasına, kullanıcıdan veri isterken ona hitaben ne söyleyeceğimizi (örneğin "Lütfen isminizi giriniz : "), karakter dizisi olarak gireriz. Bu fonksiyon çalıştırıldığında, kullanıcıya bu karakter dizisini sunarak ondan ilgili değerleri girmesini ve ENTER tuşuna basmasını bekler. Değerler girilip bu tuşa basıldıktan sonra fonksiyon, girilen bu değeri döndürür (Fonksiyonun sol tarafında "=" işareti, onun da solunda bir değişken varsa fonksiyonun döndürdüğü değer, yani kullanıcının girdiği değer, bu değişkene atılır.).

Aşağıdaki örnekte betik dosyası kullanarak kullanıcıdan isim bilgisinin alınmasını ve kullanıcıya "Merhaba *[kullanıcı ismi]*, hoş geldin." mesajının sunulmasını içeren bir gösterim yer almaktadır. Betik dosyasının içeriği şu şekildedir:

```
isim = raw_input("Lütfen isminizi giriniz : ")
print "Merhaba ", isim, ", hoş geldin."
```

Bu betik dosyası çalıştırıldığında, Python Shell ekranında, kullanıcıdan isim bilgisi istenecek ve kullanıcı bilgi girişi yapıp "ENTER" tuşuna bastıktan sonra ekrana "Merhaba *[kullanıcı ismi]*, hoş geldin." mesajı yazılacaktır. Betik dosyasının çalıştırılması ile ilgili örneği aşağıdaki kutuda bulabiliriz:

```
Lütfen isminizi giriniz : Emre
Merhaba  Emre , hoş geldin.
```

Çevre ve Alan Hesapları

Python yorumlayıcısını hesap makinesi gibi kullanabileceğimizi ve değişken tanımlama işlemini daha önceki başlıklarda incelemiştik. Bu bölümde ise, öğrenmiş olduğumuz bu iki temel bilgiyi birleştirerek çevre ve alan hesabı gibi geometri problemlerinde kullanmayı öğreneceğiz.

Python' da toplama, çıkarma, çarpma ve bölme işlemleri sırasıyla "+", "-", "*" ve "/" işaretleri ile yapılır. Bunların yanında üs alma ve kök alma işlemleri de bizler için gerekli olacaktır. Python dilinde bir **a** sayısının **b.** dereceden kuvvetini hesaplamak için, yani a^b sayısını hesaplamak için (**a** ve **b** sayılarının, aynı isimdeki "a" ve "b" değişkenlerinin içerisinde bulunduğunu düşünürsek) "**a**b**" ifadesini kullanırız. Aşağıdaki örnekte, birkaç üslü ve köklü ifadenin hesaplanması gösterilmiştir:

$$7^2 = ?\ ,\ 3^5 = ?\ ,\ \sqrt{25} = ?\ ,\ \sqrt[4]{81} = ?\ ,\ \sqrt[3]{125} = ?$$

```
>>> 7**2
49
>>> 3**5
243
>>> 25**(1/2)
1
>>> 25**(1.0/2.0)
5.0
>>> 25**0.5
5.0
>>> 81**(1.0/4.0)
3.0
>>> 81**0.25
3.0
>>> 125**(1.0/3.0)
4.9999999999999991
>>> 125**0.3
4.2566996126039225
>>> 125**0.333
4.991959282691119
>>> 125**0.33333333333
4.9999999999195275
>>> round(125**(1.0/3.0))
5.0
```

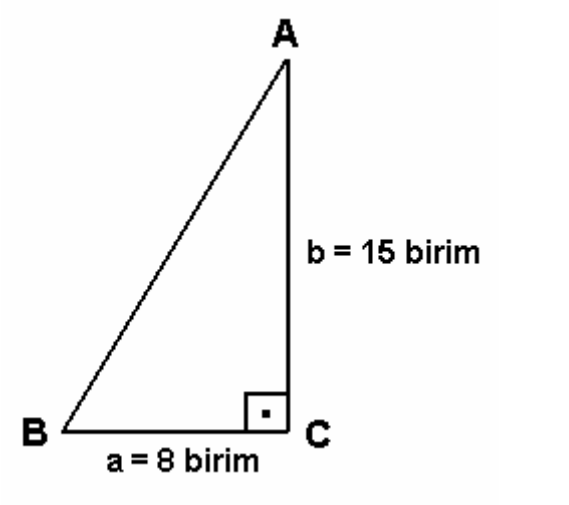
Örnekte verilen "7' nin 2. kuvveti" ve "3' ün 5. kuvveti" değerlerini bulmak için, bir önceki paragrafta yer alan "**a**b**" şeklindeki kullanım yeterli olmaktadır. Bir sayının **n.** dereceden

kökünü almak ise **(1/n).** dereceden üssünü almakla aynı olup; bu durum bize "**a**b**" şeklindeki kullanım ile yalnızca üs değil, kök de alma imkânı vermektedir.

Ancak, kök alırken üsse, yani "******" işaretlerinin sağ tarafına yazılan sayıya dikkat etmemiz gerekir. Örneğin, **25** sayısının karekökünü almak için "`25**(1/2)`" yazarsak, sonucun **5** olmasını beklememize rağmen **1** olduğunu görürüz. Bunun nedeni, üs olan "**1/2**" de yer alan **1** ve **2** sayılarının birer tamsayı olması, Python yorumlayıcısının ise **1**' i **2**' ye bölerken ondalıklı bölme yerine tam sayı bölmesi yaparak "**1' in içerisinde 2, sıfır kere vardır.**" sonucuna ulaşması ve neticede bize **25 sayısının 0. kuvvetini** döndürmesidir. Python' un ondalıklı bölme yapmasını sağlamak içinse 'üs' teki sayıları ondalıklı olarak girmemiz gerekmektedir ("**1.0/2.0**" gibi). Bunu yaptığımızda bölme işleminin sonucu **0.5** olacağından, **25** sayısının karekökü de **5.0** olarak bulunur (Gördüğümüz gibi, sonuç da bir ondalıklı sayıdır.). Bölme işleminin yerine, bölmenin sonucunu direkt olarak yazmamız da bizi doğru sonuca ulaştıracaktır ("`25**0.5`" gibi). Benzer durum, **81** sayısının **4.** dereceden kökünü alırken de geçerli olacaktır.

125 sayısının **3.** dereceden kökünü alma işlemini incelediğimizde, **125**' in üssünde yer alması gereken "**1.0/3.0**" sayısının, **0.33333333…** şeklinde sonsuza kadar devam ettiğini görürüz. Oysa ki **125** sayısının küp kökü **5**' tir. Örnekte de görüldüğü üzere, 'üs' te "**0.**" dan sonra ne kadar çok **3** koyarsak bölme işleminin sonucunu o kadar daha sağlıklı (gerçeğe yakın) hesaplamış olacağımız için küp kök işleminin sonucu da olması gereken **5** değerine o kadar çok yaklaşacaktır (Ancak ulaşamayacaktır.). Buna bir çözüm olarak, böylesi küsuratlı sayıları en yakın tamsayıya yuvarlamakta kullanılan "**round()**" fonksiyonundan yararlanarak, sonucunu yuvarlamak istediğimiz ifadeyi bu fonksiyonun parantezleri içerisine yazabiliriz. Örnekte de görüldüğü gibi bu fonksiyon aracılığı ile **125** sayısının küp kökü **5.0** olarak bulunmuştur.

Sıradaki örneğimizde ise birbirine dik olan **a** ve **b** kenarlarının uzunlukları verilmiş bir ABC üçgeninin çevresinin hesaplanması üzerinde duracağız:



Problemin çözümünde, bir dik üçgende birbirine dik olan kenarların uzunluklarının kareleri toplamının karekökünün, dik açının karşısındaki kenarın uzunluğuna eşit olduğu bilgisinden faydalanabiliriz. Dik açının karşısındaki kenara "**c**" dersek **c**' nin uzunluğu:

$|c| = \sqrt{|a|^2 + |b|^2} = \sqrt{8^2 + 15^2} = \sqrt{289} = 17$

birim olarak bulunur. Üçgenin çevre uzunluğu ise $Ç = |a| + |b| + |c| = 8 + 15 + 17 = 40$ birim olarak bulunur. Python' u etkileşimli biçimde kullanarak bu problemi, aşağıdaki şekilde çözebiliriz:

```
>>> a = 8
>>> b = 15
>>> c = (a**2 + b**2)**0.5
>>> cevre = a + b + c
print "Üçgenin çevre uzunluğu", cevre, "birimdir."
Üçgenin çevre uzunluğu 40.0 birimdir.
```

Burada dikkat etmemiz gereken şey, kenar uzunluklarını değişkenlerde tutup bu değişkenler üzerinde işlemler yaparak problem çözmenin, sadece sayılar üzerinde işlem yaparak ("`cevre=8+15+(8**2+15**2)**0.5`" gibi) sonuca varmaya kıyasla çok daha anlaşılır ve basit olduğudur.

Alıştırmalar

Alıştırma – 1

Görev

Bir betik dosyası oluşturarak içerisine, Python Shell ekranına "**Merhaba Dünya**" yazısını yazdıracak kodu uygun biçimde yazınız, kendi isminizle (Türkçe karakter ve boşluk kullanmadan) kaydediniz ve çalıştırarak sonucu gözlemleyiniz.

İpucu

Python programlama dilinde ekrana "Programlamayı seviyorum." yazdırabilmek için, aşağıdaki kod kullanılabilir:

```
print 'Programlamayı seviyorum.'
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız.

Alıştırma – 2

Görev

Bir önceki alıştırmada **print** komutunu kullanmıştık. Ekrana yazı yazdırmak için, **print** komutundan sonra bir boşluk bırakarak tek tırnak içerisine yazdırmak istediğimiz metni yazıp, tırnağı kapatmıştık.

Betik dosyası kullanmadan (Python Shell komut satırına), ekrana "**Python' u seviyorum.**" cümlesini yazdıracak bir kod yazınız. Yazacağınız komut ve size döndürülecek olan sonuç, aşağıdaki gibi olmalıdır:

```
>>> print yazacağınız kısım
Python' u seviyorum.
```

Benzer biçimde, ekrana "**Python programlama dilinin adı "piton" yılanından gelmez.**" cümlesini yazdıracak bir kod yazınız. Yazacağınız komut ve size döndürülecek olan sonuç, aşağıdaki gibi olmalıdır:

```
>>> print yazacağınız kısım
Python programlama dilinin adı "piton" yılanından gelmez.
```

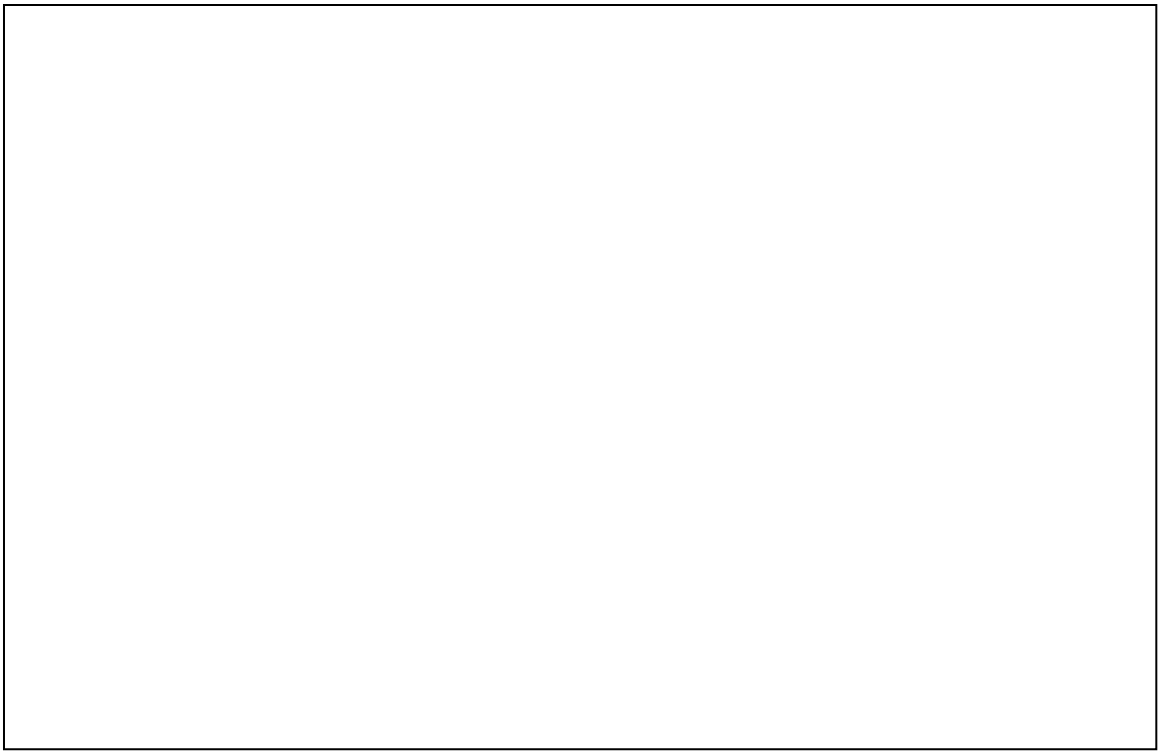
UYARI: Her iki işlemde de sonuçların, **tek tırnaklar ve çift tırnaklar da dâhil**, kutular içerisinde belirtilenlerle **aynı** olması gerekmektedir.

İpucu

[2]' de yer alan Python kılavuzlarından, kullandığınız Python sürümüne uygun olanı indirerek içerisinde "**print**" komutunun açıklandığı bölümleri incelemeniz size yardımcı olacaktır.

Sonuç

Gözleminizin sonucunu ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız. Bu konuda birden fazla çözüm olup – olmadığını belirterek birden fazla çözüm olması durumunda alternatif çözümleri de gösteriniz.



Alıştırma – 3

Görev

Python yorumlayıcısını etkileşimli biçimde kullanarak, şu problemi çözünüz:

Eğer 10 kilometrelik bir yarışı 43 dakika 30 saniyede tamamladıysanız, 1 mil mesafeyi ortalama ne kadar sürede katetmiş olursunuz? (Not: 1 mil, 1.61 kilometreye karşılık gelmektedir.)

İpucu

Python yorumlayıcısını etkileşimli bir biçimde, bir hesap makinesi gibi kullanabilirsiniz. Python' un matematiksel işlemler için kullandığı sözdizimi, standart matematiksel gösterimle hemen hemen aynıdır. Aşağıdaki iki kutuda iki farklı işlem ve bunlara ait sonuçlar yer almaktadır:

```
>>> ( 8 + 4 / 2 ) + ( 6 - 2 * 2 )
12
```

```
>>> ( ( 8 + 4 ) / 2 ) + ( ( 6 - 2 ) * 2 )
14
```

Sonuç

Bu problemi çözmek için kullanmış olduğunuz ifadeyi ve açıklamasını, varsa karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız.

Alıştırma – 4

Görev

Python yorumlayıcısını etkileşimli biçimde kullanarak aşağıdaki kutuda verilen ifadeler arasındaki farklılıkları bulunuz.

```
a = 6
b = 8.5
c = "6"
d = "8.5"
```

İpucu

"Değişken Tanımlama" bölümünü inceleyiniz.

Sonuç

Gözleminizin sonucunu ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız.

Alıştırma – 5

Görev

Aşağıdaki kutuda yer alan, Python yorumlayıcısı etkileşimli biçimde kullanılarak yazılmış ifadeleri inceleyiniz ve yorumlayınız.

```
>>> m = 8
>>> type(m)
<type 'int'>
>>> n = 12.7
>>> type(n)
<type 'float'>
>>> p = m + n
>>> type(p)
<type 'float'>
>>> print m
8
>>> print n
12.7
>>> print p
20.7
```

İpucu

"Değişken Tanımlama" bölümünü inceleyiniz.

Sonuç

Yorumlarınızı aşağıdaki kutunun içerisine yazınız.

Alıştırma – 6

Görev

Bir betik dosyası oluşturarak içerisine kullanıcıdan isim, soy isim, doğum yeri, doğum yılı ve meslek unvanı bilgilerini alıp ekrana birinci satırda "*[doğum yılı]* senesinde dünyaya gelen *[kullanıcı ismi] [kullanıcı soy ismi]*, *[doğum yeri]* doğumludur.", ikinci satırda ise "*[kullanıcı soy ismi]*, *[meslek unvanı]* olarak görev yapmaktadır." yazdıracak olan Python kodunu yazınız.

Program çalıştırıldığında kullanıcıya soracağı sorular, kullanıcının vereceği cevaplar ve programın ürettiği ekran çıktısı, aşağıdaki kutuda yer alan örnekteki gibi olmalıdır:

```
Lütfen isminizi giriniz : Emre
Lütfen soy isminizi giriniz : Gürbüz
Lütfen doğum yerinizi giriniz : Zonguldak
Lütfen doğum yılınızı giriniz : 1986
Lütfen meslek unvanınızı giriniz : bilgisayar mühendisi
1986 senesinde dünyaya gelen Emre Gürbüz , Zonguldak doğumludur.
Gürbüz , bilgisayar mühendisi olarak görev yapmaktadır.
```

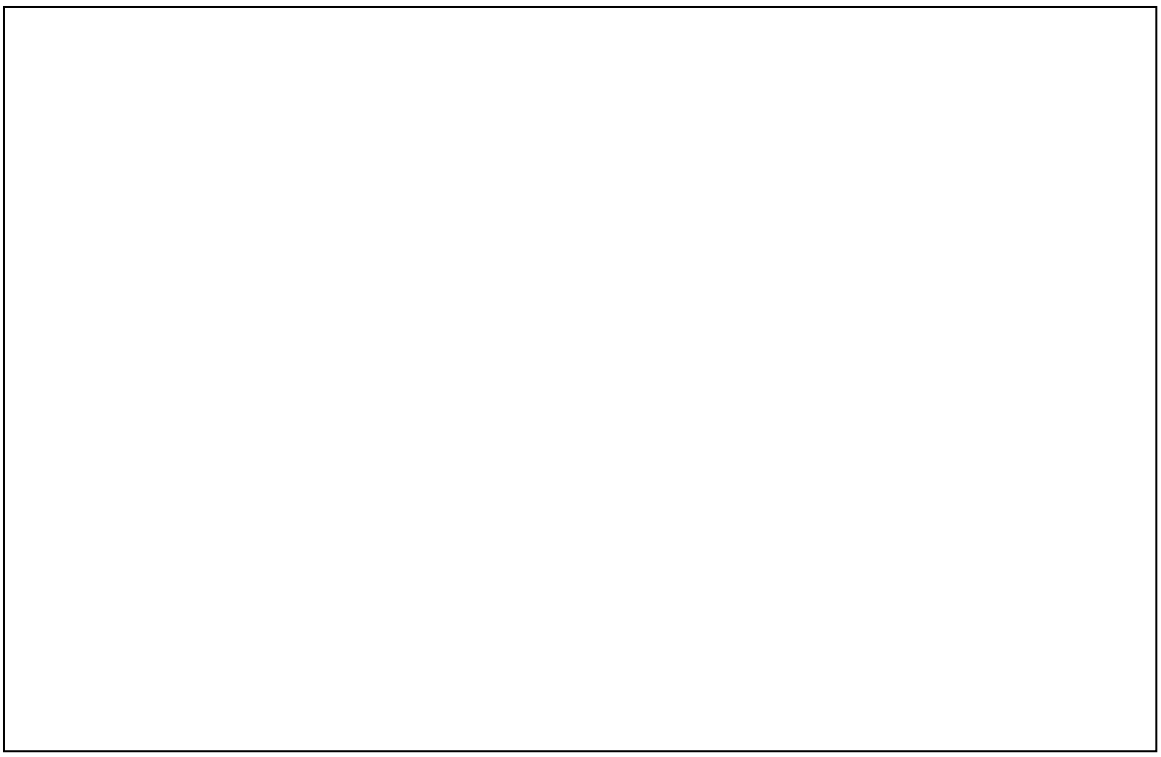
İpucu

print komutu ile tırnak içerisinde karakter dizisi yazdırırken bir alt satıra geçmek için; "**\n**" kullanılabilir, ya da iki ayrı **print** komutu kullanılabilir (Her **print** komutu yeni bir satıra yazı yazacaktır.). Örnek:

```
>>> print "Ondokuz Mayıs Üniversitesi\nSAMSUN"
Ondokuz Mayıs Üniversitesi
SAMSUN
```

Sonuç

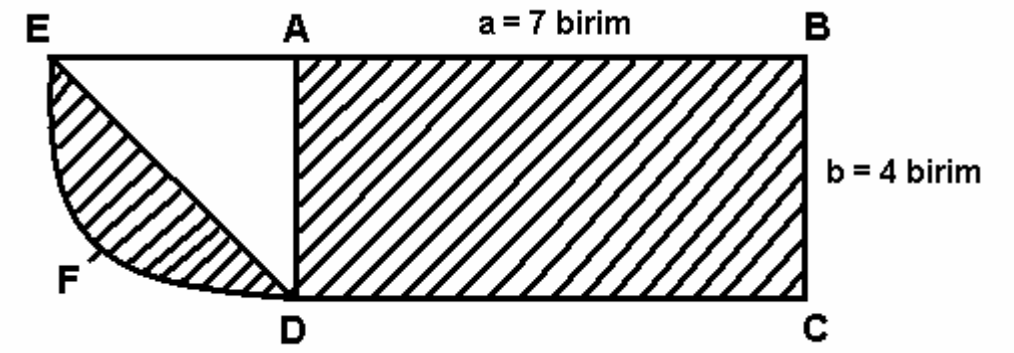
Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız.



Alıştırma – 7

Görev

Aşağıdaki şekilde ABCD bir dikdörtgen, |AB| = |CD|, |AD| = |BC|; ADE ikizkenar dik üçgen, |AD| = |AE|; DFE yayı ise merkezi A noktası olan bir çembere ait olan 90^o lik bir yay parçasıdır. ABCD dikdörtgeninin yatay kenarları 7 birim, dikey kenarları ise 4 birim uzunluğunda olduğuna göre, şekildeki taralı alanların toplamı kaç birim karedir ($\pi = 3.14$ alınız.) ?



Yukarıdaki problemi, Python yorumlayıcısını etkileşimli biçimde kullanarak "Çevre ve Alan Hesapları" başlığı altında verilen örnektekine benzer biçimde çözünüz.

Sonuç

Çözümünüz hakkındaki açıklamaları ve Python kodlarınızı aşağıdaki kutunun içerisine yazınız.

Kaynaklar

[1] http://www.python.org/doc/

[2] http://www.istihza.com/

LABORATUVAR ÇALIŞMASI – 2

Bu Çalışmanın Amacı

Bu çalışmadaki amacımız, Python' da fonksiyon kullanımını incelemek ve bu konuda çeşitli uygulamalar yapmaktır. Daha sonraki aşamada ise ara yüz tasarımı konusu ele alınacaktır.

Fonksiyonlar

Bir programlama terimi olarak fonksiyon, "bir işlemi yerine getirmek üzere bir araya getirilmiş komutlar dizisi" olarak tanımlanabilir.

Programlamada fonksiyonlar önce tanımlanırlar. Tanımlanma esnasında fonksiyonun ne şekilde çağrılacağı (argüman alıp - almayacağı), hangi işlemleri gerçekleştireceği ve bir sonuç üretip – üretmeyeceği belirlenir. Çağırma işlemi ise, fonksiyonun tanımlamasına uygun olmalıdır. Fonksiyonları argüman alıp – almamasına ve değer döndürüp – döndürmemesine göre incelersek, şu örneklere bakmamız faydalı olacaktır:

- Argüman alan ve değer döndüren fonksiyon → *Arkadaşınıza, bozması için 1 TL veriyorsunuz ve o da size iki tane 50 Kuruş veriyor.*
- Argüman alan ancak değer döndürmeyen fonksiyon → *Arkadaşınıza önceden 1 TL borcunuz vardı, ödemek için 1 TL veriyorsunuz ama ondan herhangi bir şey almıyorsunuz.*
- Argüman almayan ancak değer döndüren fonksiyon → *Arkadaşınıza bir şey vermeden ondan 1 TL borç istiyorsunuz ve size 1 TL veriyor.*
- Argüman almayan ve değer döndürmeyen fonksiyon → *Arkadaşınıza önceden 1 TL borcunuz vardı, ondan bu borcu silmesini istiyorsunuz ve o da artık kendisine borçlu olmadığınızı söylüyor. Herhangi bir şey alıp – vermiyorsunuz.*

Yukarıdaki örneklerde bahsedilen paraları **argüman**, arkadaşınıza söylemiş olduğunuz "Bu parayı bozar mısın?", "Sana borcum vardı, bu parayı alır mısın?", "Bana borç verir misin?" ve "Sana olan borcumu siler misin?" ifadelerini ise **fonksiyon çağrısı** olarak düşünebiliriz.

Fonksiyon Çağrıları

Fonksiyon çağrısı, **fonksiyonun ismini** ve ismin sağ tarafında yer alan parantezlerin içerisine **fonksiyonun aldığı argümanları** sırasıyla yazarak yapılır (Eğer fonksiyon argüman almıyorsa parantezlerin içi boş bırakılır.). Bu fonksiyon değer döndüren bir fonksiyon ise, fonksiyonun isminin sol tarafına "=" işareti, onun da soluna herhangi isimde bir değişken konularak, fonksiyonun döndürdüğü değerin, bu değişkenin içerisine atılması sağlanmış olur. Değer döndüren bir fonksiyonun sol tarafına "=" işareti ve değişken koyarak atama işlemi yapmazsak, **fonksiyonun döndürdüğü değeri ihmal etmiş oluruz**. Bu durumda fonksiyonun döndürdüğü değer de ekrana yazılacaktır. Aşağıdaki örneği inceleyerek bunu somutlaştırabiliriz:

```
>>> a = '77'
>>> type(a)
<type 'str'>
>>> deger = int(a)
>>> deger
77
>>> int(a)
77
```

Bu örnekte "a" değişkeninin içerisine, karakter dizisi olarak "'77'" değeri atılmıştır. Python' da, aldığı değerleri tamsayıya dönüştüren (dönüştürememesi durumunda uyarı veren) "**int()**" fonksiyonu var olup, değer olarak dönüşümün sonucunu döndürmektedir. Yukarıdaki kutuda dördüncü satıra baktığımız zaman "**int()**" fonksiyonunun, "a" yı tamsayıya dönüştürerek "deger" değişkeninin içerisine attığını görürüz. Bu durumda ekrana herhangi bir şey yazılmayacaktır ve "deger" isimli değişkenin değeri tamsayı olarak 77 olacaktır. Yedinci satırda ise "**int()**" fonksiyonunun döndürdüğü değer herhangi bir değişkenin içerisine atılmadığından ihmal edilmiştir. Bu durumda ise Python yorumlayıcısı dönen bu değeri ekrana yazdıracaktır.

Yukarıdaki paragrafta incelemiş olduğumuz ve bir **tür dönüştürme fonksiyonu** olan "**int()**" fonksiyonundan başka "**float()**" ve "**str()**" fonksiyonlarından da bahsedebiliriz. "**float()**" fonksiyonu da parantezleri arasına girilen değeri ondalıklı sayıya dönüştürmeyi dener, başarılı olursa bu dönüşümün sonucunu döndürür, başarısız olursa hata verir. "**str()**" fonksiyonu ise aldığı değeri karakter dizisine dönüştürür. **Bütün sayıların, harflerin ve işaretlerin birer karakter karşılığı olduğu için "str()" fonksiyonu her durumda başarılı olur.** Örneğin,

“'j'” karakterini bir tamsayıya ya da ondalıklı sayıya dönüştürmek mümkün değilken; gerek “3.277” ondalıklı sayısını, gerekse “814578” tamsayısını bir karakter dizisine dönüştürmek mümkün olabilmektedir. Aşağıdaki örneği inceleyelim:

```
>>> a = 'karakter'
>>> b = 500
>>> c = 12.34
>>> d = '500'
>>> e = '12.34'
>>> int(a)
***Hata Mesajı***
>>> float(a)
***Hata Mesajı***
>>> str(a)
'karakter'
>>> int(b)
500
>>> float(b)
500.0
>>> str(b)
'500'
>>> int(c)
12
>>> float(c)
12.34
>>> str(c)
'12.34'
>>> int(d)
500
>>> float(d)
500.0
>>> str(d)
'500'
>>> int(e)
***Hata Mesajı***
>>> float(e)
12.34
>>> str(e)
'12.34'
```

İçerisinde karakter bulunan değerlerin tamsayıya ya da ondalıklı sayıya dönüştürülmesi sırasında hata ile karşılaşmamızın nedeni, karakterlerin sayı değeri taşımamasıdır. “**int()**” fonksiyonu ise ondalıklı bir sayıyı tamsayıya dönüştürürken yuvarlama yapmasına karşın, ondalıklı sayı formatında yazılmış olan karakter dizilerinin tamsayıya dönüştürülmesini desteklememektedir. Diğer dönüşümlerde ise sorun bulunmamaktadır.

Matematiksel Fonksiyonlar

Python' da matematiksel fonksiyonları içerisinde bulunduran "math" isminde bir modül (birbiriyle ilişkili fonksiyonların bir arada toplandığı dosya) yer almaktadır. Matematiksel fonksiyonları kullanmak içinse bu modülden yararlanmamız gerekir. Python' da bir modülü çağırarak içerdiği fonksiyonları kullanabilmek için "**import**" komutu kullanılmaktadır.

Aşağıdaki örnekte, **math** modülünün ve içerdiği bazı fonksiyonların kullanımı ele alınmıştır:

```
>>> import math
>>> math.pow(5,3)
125.0
>>> math.sqrt(289)
17.0
>>> math.pi
3.1415926535897931
>>> derece = 30
>>> radyan = derece / 360.0 * 2 * math.pi
>>> radyan
0.52359877559829882
>>> math.sin(radyan)
0.49999999999999994
>>> math.sqrt(math.pow(11, 2) + math.pow(60, 2))
61.0
```

Örneğin ilk satırında, geri kalan satırlarda yer alan fonksiyonları kullanabilmemiz için gerekli olan **math** modülü çağrılmıştır. Örnekte; üs almakta kullanılan **pow** fonksiyonu, karekök almakta kullanılan **sqrt** fonksiyonu, Pi sabitini veren **pi** değişleni (math modülü içerisinde yer alan ve bu modülü çağırarak ulaşabileceğimiz sabit bir sayı) ve de radyan cinsinden bir açının sinüs değerini hesaplamakta kullanılan **sin** fonksiyonu yer almaktadır (Diğer fonksiyonlar hakkında bilgi almak için Python Shell komut satırına "`help('math')`" yazabiliriz.). En sondaki iki satırda ise bu fonksiyonların iç içe kullanılabileceğine dair bir gösterim bulunmaktadır.

Fonksiyon Tanımlama ve Kullanma

Python' da bir fonksiyon tanımlamak için "**def**" komutu kullanılır. Bir betik dosyası oluşturarak fonksiyon tanımlamak için (**import** komutunu kullanarak gerekli modülleri çağırdıktan sonra) ilk başta "**def**" yazıp bir karakter boşluk bırakarak fonksiyonumuza vermek istediğimiz ismi yazarız ve ismin hemen sağına "**(**" işaretini koyarak fonksiyonun almasını

istediğimiz parametreleri (aralarına virgül koyarak) yazıp "**)**" işaretini ve son olarak da "**:**" işaretini ekleriz. İlk satırı bu şekilde oluşturduktan sonra alt satırlara geçerek fonksiyonun ne yapmasını istiyorsak bunları satır satır kodlarız. Eğer fonksiyonun değer döndürmesini istiyorsak dosyanın en altına "**return**" komutundan sonra bir karakter boşluk bırakarak yanına döndürmek istediğimiz değişkeni ya da değeri yazarız. Anlatılanları somutlaştırmak için aşağıdaki **örnek betik dosyası**nı inceleyebiliriz:

```
def Ogrenci(isim, soyisim, numara, universite, program):
    print isim, soyisim, 'adlı', numara, 'numaralı öğrenci,',
    universite, program, 'programında eğitim görmektedir.'
```

NOT: "print" komutundan sonrasını tek bir satıra yazmalıyız.

Bu betik dosyasını kaydedip **F5** tuşuna bastıktan sonra (ya da **Run → Run Module** dedikten sonra) Python Shell ekranına yazacağımız komutlar için aşağıdaki örnek incelenebilir:

```
>>> Ogrenci('Emre', 'Gürbüz', 9210148, 'OMÜ', 'Yüksek Lisans')
Emre Gürbüz adlı 9210148 numaralı öğrenci, OMÜ Yüksek Lisans
programında eğitim görmektedir.
>>> Ogrenci('Emre', 'Gürbüz', '9210148', 'OMÜ', 'Yüksek Lisans')
Emre Gürbüz adlı 9210148 numaralı öğrenci, OMÜ Yüksek Lisans
programında eğitim görmektedir.
```

Fonksiyonun içerisinde yer alan **print** komutu, sayıları ve karakter dizilerini aynı biçimde ekrana yazdırdığı için üçüncü parametreyi sayı olarak ya da karakter dizisi olarak vermemizin bir farkı olmayacaktır.

If Yapısı

Eğer Python' da program yazarken "Bu değişkenin değeri şöyle ise programımız şöyle yapsın, değilse de böyle yapsın." şeklinde bir şartlı yönlendirmeye ihtiyaç duyuyorsak, "**if**" yapısını kullanmamız gerekir. **if** komutu, sağında yer alan kriterin doğruluğunu kontrol eder, eğer doğru ise kendi kapsamına giren (Python' a göre kendisinin altında ve girinti olarak sağda yer alan) kod parçalarının çalıştırılmasına müsaade eder. Eğer kriter doğru değilse, programın akışını, kendi kapsamında olan kod parçalarının sona erdiği yerin ilerisine atlatır (Yani kendi kapsamındaki kodların çalışmasına müsaade etmez.). Eğer burada "**else**" ifadesi var ise bu, "**if** kriteri sağlandıysa **else**' nin kapsamına giren kod parçaları es geçilecek, ama eğer **if** kriteri sağlanmadığı için program akışı **else**' nin olduğu satıra geldiyse **else**' nin kapsamında olan

kod parçaları çalıştırılacak.” anlamına gelmektedir. “**elif**” ifadesi ise **else**’ nin kritere sahip olanıdır. **elif**’ ten farklı olarak **else**’ de bir de (tıpkı **if**’ te olduğu gibi) sağ tarafta yer alan karşılaştırma kriterinin doğruluğu kontrol edilir ve doğruluk sağlanırsa **elif**’ in kapsamında bulunan kod parçaları çalıştırılır.

if yapısında kontrol edilebilecek kriterler; eşitlik, farklılık, büyüklük, küçüklük, büyük-eşitlik ve küçük-eşitliktir. Konuyu daha anlaşılır kılmak adına, betik dosyasında yer alan aşağıdaki fonksiyon örneğini inceleyerek yorumlamamız faydalı olacaktır:

```
def YasBoyKilo(yas, boy, kilo):
    boyKusurat = 100.0 * float(boy) - 100.0
    fark = boyKusurat - kilo

    if yas < 30:
        if fark > 10.0:
            print 'Zayıfsınız, enerji veren yiyecekler tüketmeniz önerilir.'
        elif (10.0 >= fark) & (fark >=-10.0):
            print 'Kilonuz normal.'
        elif -10.0 > fark:
            print 'Kilolusunuz, spor yapmanız önerilir.'
    else:
        if fark > 10.0:
            print 'Zayıfsınız, yorucu hareketlerden kaçınmanız önerilir.'
        elif (10.0 >= fark) & (fark >=-10.0):
            print 'Yaşınıza rağmen formdasınız, tebrikler...'
        elif -10.0 > fark:
            print 'Kilolusunuz, diyet yapmanız önerilir.'
```

NOT: 8. ve 15. satırlardaki “&” işareti “ve” bağlacı olup, sağındaki ve solundaki kriterlerin aynı anda sağlanmasını zorunlu kılar. Eğer bu kriterlerden en az biri sağlanmıyorsa **elif** komutunun sağındaki tüm kriterler sağlanmamış sayılırlar.

Bu fonksiyondaki şartlı yapıyı şu şekilde açıklayabiliriz:

Eğer **yas** 30’ dan küçükse **fark**’ a bak. Eğer **fark** 10’ dan büyükse ekrana “Zayıfsınız, enerji veren yiyecekler tüketmeniz önerilir.” yaz, **fark** 10 ile -10 arasında ise ekrana “Kilonuz normal.” yaz, **fark** -10’ dan da küçükse ekrana “Kilolusunuz, spor yapmanız önerilir.” yaz. Eğer **yas** 30’ dan küçük değilse (30 ise ya da 30’ dan büyük ise) gene **fark**’ a bak. Eğer **fark** 10’ dan büyükse ekrana “Zayıfsınız, yorucu hareketlerden kaçınmanız önerilir.” yaz, **fark** 10 ile -10 arasında ise ekrana “Yaşınıza rağmen formdasınız, tebrikler…” yaz, **fark** -10’ dan da küçükse ekrana “Kilolusunuz, diyet yapmanız önerilir.” yaz.

Bu fonksiyonu bir betik dosyasına kaydedip **F5** tuşuna basarak (ya da **Run → Run Module** seçimini yaparak) Python Shell ekranına döndükten sonra çalıştırılmasını inceleyelim:

```
>>> YasBoyKilo(28, 1.70, 72)
Kilonuz normal.
>>> YasBoyKilo(20, 1.80, 55)
Zayıfsınız, enerji veren yiyecekler tüketmeniz önerilir.
>>> YasBoyKilo(18, 1.75, 120)
Kilolusunuz, spor yapmanız önerilir.
>>> YasBoyKilo(75, 1.70, 52)
Zayıfsınız, yorucu hareketlerden kaçınmanız önerilir.
>>> YasBoyKilo(48, 1.54, 51)
Yaşınıza rağmen formdasınız, tebrikler...
>>> YasBoyKilo(63, 1.77, 108)
Kilolusunuz, diyet yapmanız önerilir.
```

While Döngüsü

Programlama yaparken "Şu şart sağlandığı sürece şu işlemleri yap, şart sağlanmadığı andan itibarense yapmayı durdur ve bir daha yapma." şeklinde bir kod tasarımına gereksinim duyuyorsak "**while**" yapısını kullanabiliriz. **while** komutu da tıpkı **if** komutu gibi kendi sağında yer alan kriterin geçerliliğini kontrol eder ve kriter geçerli ise kendi kapsamındaki kodları (altında ve girinti olarak kendisinden daha sağda yer alan kod parçalarını) 1 kez çalıştırır. Bu çalıştırma esnasında, kriter içerisinde kıyaslama yaptığı değişkenlerin değerinde değişiklik olabilir. Çalıştırma sona erdikten sonra döngü tekrar başa döner ve kriterin geçerliliği tekrar kontrol edilir ve geçerli ise aynı kod parçaları tekrardan çalıştırılır. Bu işlemler, kriter geçersiz kalana kadar tekrarlanır, kriter geçersiz kaldıktan sonra bir daha tekrarlanmaz. Anlatımı somutlaştırmak için aşağıdaki örneği inceleyebiliriz:

```
def DikdortgenCiz(en, boy):
    isaret = ''
    while en > 0:
        isaret = isaret + '*'
        en = en - 1
    while boy > 0:
        print isaret
        boy = boy - 1
```

Betik dosyası kullanılarak çalıştırılmak üzere hazırlanmış ve dikdörtgen çizmeye yarayan bu fonksiyonda kullanıcıdan dikdörtgenin boyu ve eni alınmakta ve bu değerlere uygun bir dikdörtgen çizdirilmektedir.

Üstteki **while** döngüsünde, dikdörtgenin eni kadar uzunlukta ve "*" işaretlerinden oluşan yatay bir çizgi oluşturulmaktadır. İlk başta, "işaret" adlı karakter dizisinin içi boştur. Örneğin "en" değeri 6 ise, **while**' ın olduğu satırda 6' nın 0' dan büyük olduğu ifadesi doğrulanacaktır. 6 sayısı 0' dan büyük olduğu için **while**' ın kapsamına giren satırlardaki (hemen alttaki 2 satır) kodlar çalıştırılacak, "isaret" değişkeninin içeriği "*" olacak, "en" değeri ise 5 olacaktır. Sonra programın akışı tekrar **while** kısmına gelecek, "en" değerinin 0' dan büyük olduğu ifadesi doğrulanmaya çalışılacaktır. 5 sayısı da 0' dan büyük olduğu için "isaret" değişkeninin değeri "**" olacak, "en" değeri ise 4 olacaktır. Bu işlem, "en" değeri 0' a inene kadar devam edecek, daha sonra **0 > 0** şartı sağlanamayacağı için sona erecektir. Sonuçta "isaret" değişkeninin içeriği "******" olacaktır.

Daha sonra program akışı alttaki **while** döngüsüne geçecektir. Örneğin "boy" değeri 4 ise, üsttekine benzer biçimde, bu döngü de 4 kere çalışacak, ekrana 4 kere "******" yazdırılacaktır ("isaret" değişkeninin içeriği "******" olduğu için.). **print** komutu her kullanılışında yeni bir satıra geçtiği için bu işaretler alt alta yazdırılacak, sonuç olaraksa 6x4 boyutlarında bir dikdörtgen elde edilecektir.

Örnek kullanımı inceleyelim:

```
>>> DikdortgenCiz(6, 4)
******
******
******
******
>>> DikdortgenCiz(3, 5)
***
***
***
***
***
```

For Döngüsü

"**for**" döngüsünün **while** döngüsünden tek farkı, hem kriter geçerliliği kontrolünün, hem de kriterde kullanılan değişkenlerin değerlerinin güncellenmesi işleminin aynı satırda (**for** komutunun bulunduğu satırda) yapılıyor olmasıdır. Bu konuyu örnekle açıklamak daha yararlı olacaktır. Yukarıdaki dikdörtgen çizme fonksiyonunu bu kez de **for** komutlarını kullanarak gerçekleştirelim:

```
def DikdortgenCiz(en, boy):
    isaret = ''
    for i in range(0, en):
        isaret = isaret + '*'
    for j in range(0, boy):
        print isaret
```

Üstteki **for** döngüsünde, "i" isimli bir değişken tanımlanmış (İstediğimiz ismi verebiliriz.), bu **i** değişkeni için **0** ile "**en**" değişkeninin değeri aralığında değerler belirlenmiştir. Yani, "en" değeri 6 olarak girildiyse bu, "i = 0 için bir kere çalış, daha sonra i = 1 için bir kere daha çalış, daha sonra i = 2 için…, son olarak da i = 5 için çalış" anlamına gelmektedir. **range** içerisindeki alt değerin (0) dahil, üst değerin (6) ise hariç tutulduğunu unutmamalıyız. Bu döngü tamamlandığında ise "isaret" değişkeninin içeriği "******" olacaktır.

Benzer şekilde, örneğin "boy" değeri 4 ise alttaki **for** döngüsü de 4 kez çalışacaktır. Programın örnek kullanımı ise **while** örneğindeki kullanımla aynı olacaktır:

```
>>> DikdortgenCiz(6, 4)
******
******
******
******
>>> DikdortgenCiz(3, 5)
***
***
***
***
***
```

Ara Yüz Kullanımı

Python' da birbiriyle ilişkisi olan birden çok fonksiyonu tek bir dosyada toplayarak, daha sonra bu fonksiyonlardan bir ya da birden fazlasına bu dosya üzerinden ulaşmak mümkündür. Böylesi bir kullanım, istediğimiz fonksiyona daha kolay ve hızlı bir biçimde ulaşmamızı sağlayacak, daha anlaşılır ve planlı bir programlama yapmamıza katkı sağlayacaktır.

Örnek olarak "SekilCiz.py" isminde bir betik dosyamız olsun (aşağıdaki kutuda) ve içerisinde dik üçgen çizme, dikdörtgen çizme ve kare çizme fonksiyonlarını bulundursun:

```
def DikdortgenCiz(en, boy):
    isaret = ''
    for i in range(0, en):
        isaret = isaret + '*'
    for j in range(0, boy):
        print isaret

def DikUcgenCiz(taban):
    isaret = ''
    for i in range(0, taban):
        isaret = isaret + '*'
        print isaret

def KareCiz(kenar):
    isaret = ''
    for i in range(0, kenar):
        isaret = isaret + '*'
    for j in range(0, kenar):
        print isaret
```

Biz de bir betik dosyası oluşturarak sadece dörtgen çizimi yapan ve diğer şekillerin çizimleriyle ilgilenmeyen bir tasarım yapalım. Oluşturacağımız yeni betik dosyasına "DortgenCiz.py" ismini verelim ve "SekilCiz.py" içerisinde yer alan ve dörtgenlerle ilgili olan fonksiyonları **tekrar yazmadan**, sadece çağırarak kullanalım. "SekilCiz.py" dosyasındaki fonksiyonların prototiplerini (isimleri ve argümanları) öğrenmek için Python Shell ekranında "help('SekilCiz')" yazabiliriz:

```
>>> help('SekilCiz')
Help on module SekilCiz:

NAME
    SekilCiz

FILE
    c:\python25\sekilciz.py

FUNCTIONS
    DikUcgenCiz(taban)

    DikdortgenCiz(en, boy)

    KareCiz(kenar)
```

"FUNCTIONS" bölümünde, "SekilCiz.py" dosyası içerisinde yer alan fonksiyonların prototiplerini görmek mümkündür. "DortgenCiz.py" dosyasını oluştururken de "SekilCiz.py" içerisindeki fonksiyonları yeniden yazmak yerine sadece çağıracağımız için, prototiplerini bilmemiz yeterli olacaktır.

"DortgenCiz.py" betik dosyasının içeriği, aşağıdaki gibi olacaktır:

```
from SekilCiz import DikdortgenCiz, KareCiz

def DortgenCiz_Dikdortgen(Dikd_en, Dikd_boy):
    DikdortgenCiz(Dikd_en, Dikd_boy)

def DortgenCiz_Kare(Dikd_kenar):
    KareCiz(Dikd_kenar)
```

Bu dosyayı açarak **F5** ile Python Shell ekranına geçtiğimizde gerçekleştirebileceğimiz örnek kullanım aşağıdadır:

```
>>> DortgenCiz_Dikdortgen(6, 4)
******
******
******
******
>>> DortgenCiz_Kare(3)
***
***
***
```

Tüm bu işlemleri değerlendirelim. Örneğin elimizde bir müzik seti var, biz de hangi tuşa basıldığında müzik CD' sini dinleyebileceğimizi, hangi düğmeyle radyoyu açabileceğimizi, hangi kontrolü kullanarak ses ayarı yapabileceğimizi biliyoruz. Ancak, bu düğmelere basıldığında müzik setinin içerisinde gerçekleşen fiziksel olayları bilmiyoruz ve ilgilenmek de

istemiyoruz, çünkü bizim amacımız sadece müzik dinlemek ve kontrollerin ne işe yaradığını bilmemiz bizim için yeterli. Müzik setinin üzerindeki düğmeler ve kontroller, bu aletin **ara yüzü**nü oluşturmaktadır.

İşte burada da, "SekilCiz.py" dosyası içerisinde yer alan fonksiyonların nasıl kullanıldığını bilip, arka planda neler yaptığını bilmeye gerek duymadan, "DortgenCiz.py" içerisinde çağırarak kullandık. "SekilCiz.py" deki fonksiyonların **prototipleri**, aynı zamanda **ara yüzleri** olarak da düşünülebilir. Bu fonksiyonları çağırabilmek için, "DortgenCiz.py" betik dosyasının ilk satırında yer alan "**from … import …**" komutunu yazmak zorunda olduğumuzu da unutmamalıyız.

NOT: "DortgenCiz.py" içerisindeki fonksiyonların isimleri ve parametreleri, "SekilCiz.py" içerisindeki fonksiyonların isim ve parametreleriyle karıştırılmaması için farklı yazılmıştır.

Değer Döndüren Fonksiyonlar

Yazdığımız fonksiyonun çeşitli işlemler ve hesaplamalar sonucunda bir değer üreterek bunu döndürmesini istiyorsak "**return**" komutunu kullanmamız gerektiğine değinmiştik. Şimdi bunu örnek kod üzerinde görelim:

```
def MutlakDeger(tamsayi):
    if tamsayi < 0:
        sonuc = tamsayi * (-1)
    else:
        sonuc = tamsayi
    return sonuc
```

Yukarıda, kendisine argüman olarak verilen tamsayıların mutlak değerini alan bir fonksiyon verilmiştir. "**tamsayi**" değerine göre şartlı işlem yapılarak her şart için hesaplanan değer "**sonuc**" değişkeni içerisine atılmakta ve fonksiyon da bu **sonuc** değişkeninin değerini döndürmektedir. Örnek kullanım:

```
>>> MutlakDeger(6)
6
>>> MutlakDeger(-9)
9
>>> MutlakDeger(0)
0
```

Alıştırmalar

Alıştırma – 1

Görev

Python' da gerek değişkenlerin, gerekse değişkenlerin içerisine atanan değerlerin **tür**lerini döndüren "**type**" fonksiyonunu daha önceki bölümlerde incelemiştik. Aşağıdaki kutuyu inceleyiniz ve "**type**" fonksiyonunu **argüman alma** ve **değer döndürme** bakımından gözlemleyiniz. Değer döndürüp–döndürmediğini, döndürüyor ise döndürdüğü değerin türünü belirtiniz.

```
>>> a = 60
>>> type(a)
<type 'int'>
>>> b = type(a)
>>> b
<type 'int'>
>>> c = type(b)
>>> c
<type 'type'>
```

İpucu

"Fonksiyonlar" ve "Fonksiyon Çağrıları" bölümlerini inceleyiniz.

Sonuç

Gözlemlerinizin sonucunu ve yorumlarınızı aşağıdaki kutunun içerisine yazınız.

Alıştırma – 2

Görev

Bir betik dosyası kullanarak ve **math** modülündeki fonksiyonlardan yararlanarak, yarıçapı verilen bir dairenin çevre uzunluğunu ve alanını hesaplayarak bunları ekrana yazdıran bir fonksiyon hazırlayınız ve Python Shell komut satırından bu fonksiyonu çalıştırınız. Fonksiyon argüman olarak yarıçap uzunluğunu almalı, herhangi bir değer döndürmemelidir. Çevre uzunluğu ve alan değerlerini hesapladıktan sonra bunları anlaşılır bir biçimde ekrana yazmalıdır (π sayısını **math** modülünden elde ediniz.).

Fonksiyonunuza "DaireFonk" ismini verdiğinizi varsayarsak bu fonksiyon Python Shell komut satırında çalıştırıldığında ekranda görülmesi beklenen sonuç aşağıdaki gibi olmalıdır:

```
>>> DaireFonk(10)
Dairenin çevresi 62.8318530718 birimdir.
Dairenin alanı 314.159265359 birim karedir.
```

İpucu

"Matematiksel Fonksiyonlar" ve "Fonksiyon Tanımlama ve Kullanma" bölümlerini inceleyiniz. Betik dosyasının en başında (fonksiyon tanımlamadan hemen önce) ihtiyaç duyacağınız modülü / modülleri çağırmayı unutmayınız.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız.

Alıştırma – 3

Görev

İkinci dereceden bir bilinmeyenli denklemler, aşağıdaki biçimde ifade edilirler:

$$ax^2 + bx + c = 0$$

İki tane kökü (w_1 ve w_2) bulunan bu denklemlerde kökleri hesaplamak için:

$$w_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad \text{ve} \qquad w_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

eşitlikleri kullanılır. Eğer

$(b^2 - 4ac) >= 0$ ise denklemin kökleri hesaplanabilir, $(b^2 - 4ac) < 0$ ise kökler karmaşık olacağı için hesaplanamaz.

Betik dosyası kullanarak, kullanıcıdan sırasıyla x^2' li terimin katsayısını, x' in katsayısını ve sabit terimi alarak denklemin köklerinin hesaplanabilirliğini kontrol ettikten sonra eğer hesaplanabiliyorsa hesaplayan ve bunları ekrana yazdıran bir fonksiyon hazırlayınız. Köklerin hesaplanamaması durumunda ise ekrana "Denklemin gerçel kökü bulunmamaktadır." bilgisinin yazdırılmasını sağlayınız. Python Shell ekranında fonksiyonun çağırılması ve kullanımı, aşağıdaki örnekteki gibi olmalıdır (fonksiyona "KokHesapla" isminin verildiğini varsayarsak):

```
>>> KokHesapla(1, -1, -6)
Denklemin birinci kökü : 3.0
Denklemin ikinci kökü  : -2.0
>>> KokHesapla(1, -2, 1)
Denklemin birinci kökü : 1.0
Denklemin ikinci kökü  : 1.0
>>> KokHesapla(7, 3, 6)
Denklemin gerçel kökü bulunmamaktadır.
```

Birinci işlemde $x^2 - x - 6 = 0$, ikinci işlemde $x^2 - 2x + 1 = 0$, üçüncü işlemde ise $7x^2 + 3x + 6 = 0$ denklemi ele alınmıştır.

İpucu

"Matematiksel Fonksiyonlar", "Fonksiyon Tanımlama ve Kullanma" ve "If Yapısı" bölümlerini inceleyiniz. Betik dosyasının en başında (fonksiyon tanımlamadan hemen önce) ihtiyaç duyacağınız modülü / modülleri çağırmayı unutmayınız.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız.

Alıştırma – 4

Görev

"**math**" modülünde yer alan "**exp**", "**pow**" ve "**sqrt**" fonksiyonlarının prototiplerinden yararlanarak, üssel fonksiyonların yer aldığı "UsselFonksiyonlar.py" isimli bir betik dosyası oluşturunuz.

NOT: Matematikte bir **e** sabit sayısı vardır ve yaklaşık değeri **2.71828**' dir. **math** modülündeki **exp(x)** fonksiyonu da e^x değerini döndürür. **math** modülündeki **pow(x,y)** fonksiyonu x^y değerini döndürürken **sqrt(x)** fonksiyonu da $\sqrt{x}$ değerini döndürür. Sizden, bu fonksiyonların prototiplerini kullanarak yeni bir betik dosyası oluşturmanız istenmektedir. **Bunu yapmaktaki amacımız, pek çok matematiksel fonksiyonun yer aldığı math sınıfından, sadece üssel olanları ayıklayarak üssel işlemler için özelleşmiş bir sınıf oluşturmaktır.**

Bu dosya **F5** ile çalıştırıldığında örnek kullanım aşağıdaki gibi olmalıdır (e^x değerini hesapladığımız fonksiyona "e_ussu", x^y değerini hesapladığımız fonksiyona "us_hesapla", $\sqrt{x}$ değerini hesapladığımız fonksiyona ise "karekok_al" isimlerini verdiğimizi farz edersek):

```
>>> e_ussu(3)
20.085536923187668
>>> us_hesapla(7, 2)
49.0
>>> karekok_al(169)
13.0
```

İpucu

"Ara Yüz Kullanımı" ve "Değer Döndüren Fonksiyonlar" bölümlerini inceleyiniz. Betik dosyasının en başında (fonksiyon tanımlamadan hemen önce) ihtiyaç duyacağınız modülü / modülleri çağırmayı unutmayınız.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız.

LABORATUVAR ÇALIŞMASI – 3

Bu Çalışmanın Amacı

Bu çalışmadaki amacımız, Python programlama dilinde yer alan operatörlerin kullanımı ve özyineleme özelliği hakkındaki bilgilerimizi pekiştirmek ve değer döndüren fonksiyonlarla ilgili uygulamalar yapmaktır.

Operatörler

Python' da, "**bool**" ismi verilen bir değişken türü vardır. **bool** türündeki bir değişkenin türü ya "**True** (doğru)", ya da "**False** (yanlış)" olabilir. Yanındaki kriter sağlandığında (değeri **True** olduğunda) kendi kapsamındaki kod parçalarının çalıştırılmasına müsaade ederken kriter sağlanmadığında (değeri **False** olduğunda) buna müsaade etmeyen **if** ve **while** yapılarında **bool** türündeki ifadeler sıklıkla kullanılırlar.

Bazı operatörler (+, -, *, /, % gibi) sonuç olarak **sayı** ya da **karakter dizisi** döndürürken bazıları da (==, !=, <, >, <=, >= gibi) **bool** tipinde değerler döndürürler.

"**and**", "**or**" ve "**xor**" mantıksal operatörleri ise değeri "**True**" ya da "**False**" olan ifadelerin arasında bulunarak bağlaç görevi görürler. Bu operatörlerden **and** ile **or** yazı ile yazılabildiği gibi, **and** operatörünü "**&**" işareti ile, **or** operatörü de "|" işaretleri ile göstermek de mümkündür. **xor** operatörü ise yalnızca "^" işareti ile kullanılabilir. Bu bağlaçların döndürecekleri sonuç aşağıdaki tablodan yararlanılarak çıkarılabilir:

SOLDAKİ DEĞER	OPERATÖR	SAĞDAKİ DEĞER	SONUÇ
True	**and** *ya da* **&**	*True*	***True***
True	**and** *ya da* **&**	*False*	***False***
False	**and** *ya da* **&**	*True*	***False***
False	**and** *ya da* **&**	*False*	***False***
True	**or** *ya da* **|**	*True*	***True***
True	**or** *ya da* **|**	*False*	***True***
False	**or** *ya da* **|**	*True*	***True***
False	**or** *ya da* **|**	*False*	***False***
True	^	*True*	***False***
True	^	*False*	***True***
False	^	*True*	***True***
False	^	*False*	***False***

"**not**" operatörü ise kendisinden sonra gelen ifadenin doğruluk değerini değiştirerek **True** ise **False**, **False** ise **True** yapar:

```
>>> 7==7
True
>>> not 7==7
False
```

Diğer operatörlerin görevlerini hatırlayalım:

+ → *İki sayıyı toplar ya da iki karakter dizisini birleştirir. Bir fonksiyon gibi işlemin sonucunu döndürür.*

```
>>> 3+9.0
12.0
>>> 'pyt'+'hon'
'python'
```

- → *İki sayının farkını alır. Bir fonksiyon gibi işlemin sonucunu döndürür.*

```
>>> 8.74-19
-10.26
```

***** → *İki sayıyı çarpar ya da bir karakter dizisini belirli bir sayıda tekrarlayarak arka arkaya ekler. Bir fonksiyon gibi işlemin sonucunu döndürür.*

```
>>> 13*61
793
>>> 'ekim'*4
'ekimekimekimekim'
```

/ → *İki sayıyı böler ve **bölüm**ü hesaplar. Bölen ve bölünen tamsayı ise kalan ihmal edilir. Bölen ve bölünenden en az biri ondalıklı sayı ise bölme işlemi kalan üzerinden de devam ederek ondalıklı bölme yapılır. Bir fonksiyon gibi işlemin sonucunu döndürür.*

```
>>> 7/4
1
>>> 7/4.0
1.75
```

% → *İki sayıyı böler ve **kalan**ı hesaplar (Başka bir deyişle bir sayının diğerine göre **mod**unu hesaplar.). Bir fonksiyon gibi işlemin sonucunu döndürür.*

```
>>> 7%4
3
>>> 7%4.0
3.0
```

== → *İki ifadenin eşitliğini kontrol ederek **True** ya da **False** döndürür.*

```
>>> 8==9
False
>>> 7=='yedi'
False
>>> '19mayis'=='19mayis'
True
>>> 35==35.0
True
>>> 'apostrof'=="apostrof"
True
```

!= → *İki ifadenin farklılığını kontrol ederek **True** ya da **False** döndürür.*

```
>>> 8!=9
True
>>> 7!='yedi'
True
>>> '19mayis'!='19mayis'
False
>>> 35!=35.0
False
>>> 'apostrof'!="apostrof"
False
```

< → *Soldaki ifadenin sağdakinden küçüklüğünü kontrol ederek **True** ya da **False** döndürür.*

```
>>> 8<9
True
>>> 4.41<4.40
False
>>> 'aa'<'a'
False
>>> 'aa'<'aaa'
True
>>> 'ab'<'aa'
False
>>> 'ab'<'ac'
True
```

> → *Soldaki ifadenin sağdakinden büyüklüğünü kontrol ederek **True** ya da **False** döndürür.*

<= → *Soldaki ifadenin sağdakinden küçük ya da sağdakine eşit olma durumunu kontrol ederek **True** ya da **False** döndürür.*

>= → *Soldaki ifadenin sağdakinden büyük ya da sağdakine eşit olma durumunu kontrol ederek **True** ya da **False** döndürür.*

Özyineleme

Özyineleme, "bir ifadenin kendi kendini tekrar etmesi", "bir fonksiyonun kendi içinde kendini çağırması" olarak tanımlanabilir. Örnek verecek olursak:

```
def ozyineleme():
    ozyineleme()
```

Yukarıdaki kodu bir betik dosyasına yazarak "ozyineleme" fonksiyonunu çağırdığımızda bu fonksiyon defalarca kendini çağıracak ve **maksimum özyineleme sayısı** aşıldığında bu iç içe çağırma işlemi sona erecektir. Fonksiyon çalıştırıldığında görülecek sonuç:

```
  File "C:/Python25/ozyineleme.py", line 2, in ozyineleme
    ozyineleme()
  File "C:/Python25/ozyineleme.py", line 2, in ozyineleme
    ozyineleme()
  File "C:/Python25/ozyineleme.py", line 2, in ozyineleme
    ozyineleme()
  ...
  ...
RuntimeError: maximum recursion depth exceeded
```

Başka bir örneği inceleyelim:

```
def gerisay(tavan):
    if tavan <= 0:
        print 'Başla!'
    else:
        print tavan
        gerisay (tavan-1)
```

Bir betik dosyasında yer alan bu fonksiyonu, "**tavan**" değerine **3** vererek çalıştıracak olursak şu sonucu elde ederiz:

```
>>> gerisay(3)
3
2
1
Başla!
```

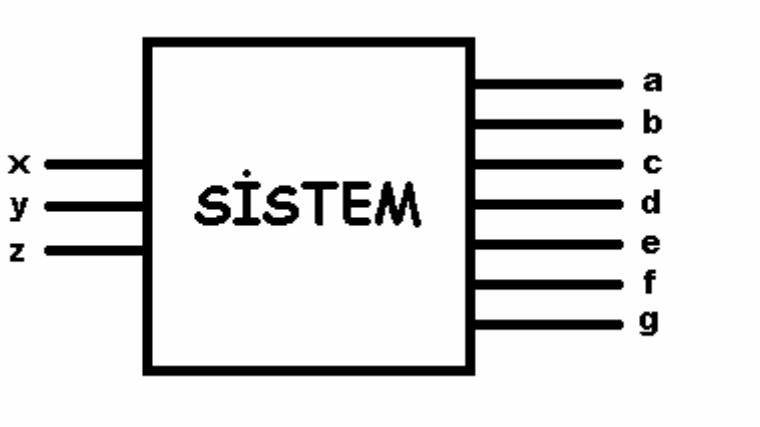
Burada ilk önce "**gerisay**" fonksiyonuna **3** değeri verilmektedir. Fonksiyon, **tavan** değeri **3** olduğu için **if** bloğuna girmeyip, **else** içerisine girecektir, **tavan** değerini (3) ekrana yazdıktan sonra, "tavan – 1 (yani **2**)" değeri ile kendini çağıracaktır. Bu satırdaki işlemi "**gerisay(2)**" olarak düşünebiliriz. Daha sonra fonksiyon, **2** değeri girilerek sil baştan çağrıldığı için aynı işlemler yeniden yapılacak, **if** bloğu atlanarak **else** içerisinde ekrana **tavan** değeri (2) yazılacaktır ve fonksiyon," tavan – 1 (yani **1**)" değeri ile çağrılacaktır. Bu işlemin sonunda da "**tavan(1)**" çağrısı yapılacak, ekrana tavan değeri olan **1** yazılacak ve en sonda "**tavan(0)**" çağrısı yapılacaktır. Ancak, **0** değeri verilerek fonksiyon tekrar çağrıldığında "**0 <= 0**" şartı sağlanacağı için bu sefer **if** bloğuna girilirken **else** bloğuna girilmeyecektir. **if** bloğuna girildiğinde ekrana "**Başla!**" yazısı yazılarak program sonlanacaktır. Programın sonlanmasının nedeni, en son olarak girmiş olduğu **if** bloğunun içerisinde kendi kendini yeniden çağırmasını söyleyen herhangi bir komut bulunmamasıdır.

Alıştırmalar

Alıştırma – 1

Görev

x, **y**, ve **z** girişleri; **a**, **b**, **c**, **d**, **e**, **f** ve **g** çıkışları olan bir sistem ve bu sisteme ait giriş–çıkışlara ait tablo aşağıda verilmiştir:



SİSTEM GİRİŞ-ÇIKIŞ TABLOSU									
GİRİŞLER			ÇIKIŞLAR						
x	y	z	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0
1	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	1

Betik dosyası kullanarak, kullanıcıdan **x**, **y** ve **z** değerlerini argüman olarak alan ve çıkış değerlerini ekrana yazdıran "**YolSecici**" isimli bir fonksiyon yazınız ve betik dosyasına "**YolSecici.py**" ismini veriniz. Mümkün olduğu kadar **az** satırda kodlama

yaparak gerçekleştirimde bulunmanız gerekmektedir. Gerçekleştirimini yapacağınız fonksiyon çalıştırıldığında elde edilecek ekran görüntüsü şu şekilde olmalıdır:

```
>>> YolSecici(0,0,0)
a : 0
b : 0
c : 0
d : 0
e : 0
f : 0
g : 0
>>> YolSecici(0,0,1)
a : 1
b : 0
c : 0
d : 0
e : 0
f : 0
g : 0
>>> YolSecici(0,1,0)
a : 0
b : 1
c : 0
d : 0
e : 0
f : 0
g : 0
>>> YolSecici(0,1,1)
a : 0
b : 0
c : 1
d : 0
e : 0
f : 0
g : 0
```

```
>>> YolSecici(1,0,0)
a : 0
b : 0
c : 0
d : 1
e : 0
f : 0
g : 0
>>> YolSecici(1,0,1)
a : 0
b : 0
c : 0
d : 0
e : 1
f : 0
g : 0
>>> YolSecici(1,1,0)
a : 0
b : 0
c : 0
d : 0
e : 0
f : 1
g : 0
>>> YolSecici(1,1,1)
a : 0
b : 0
c : 0
d : 0
e : 0
f : 0
g : 1
```

İpucu

Gerçekleştirmeniz beklenen fonksiyonun başlangıcı, aşağıdakine benzer olmalıdır:

```
def YolSecici(x, y, z):
    if x == .....
.....
```

"Operatörler" bölümünü ve bir önceki laboratuvar çalışmamızda yer alan "If Yapısı" bölümünü inceleyiniz.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız.

Alıştırma – 2

Görev

Matematikte 1' den büyük pozitif bir tamsayı ile **1** arasındaki (bahsedilen tamsayı da dahil) tüm tamsayıların çarpımına, ilgili tamsayının **faktöriyel**i denir. 0 ve 1 sayılarının faktöriyeli **1**' e eşittir. Negatif tamsayıların ise faktöriyel hesaplaması yapılamaz. Örneğin:

3' ün faktöriyeli = **3!** = 3 * 2 * 1 = **6**

7' nin faktöriyeli = **7!** = 7 * 6 * 5 * 4 * 3 * 2 * 1 = **5040**

Özyineleme kullanarak "**Faktoriyel**" isimli bir faktöriyel alma fonksiyonu yazınız ve bunu "**Faktoriyel.py**" isimli bir betik dosyasına kaydediniz. Fonksiyonunuz argüman olarak bir tamsayı alıp, değer olarak ise bu tamsayının faktöriyelini döndürmelidir. Negatif bir tamsayı verildiğinde ise ekrana uyarı olarak "**Negatif tamsayıların faktöriyel hesaplaması yapılamaz**." yazmalıdır. Sizden beklenen gerçekleştirimin çalıştırılmasına ait bir örnek gösterim aşağıdaki gibidir:

```
>>> Faktoriyel(6)
720
>>> Faktoriyel(-4)
Negatif tamsayıların faktöriyel hesaplaması yapılamaz.
>>> Faktoriyel(0)
1
>>> Faktoriyel(1)
1
```

İpucu

"Özyineleme" bölümünü inceleyiniz. Değer döndüren bir fonksiyon içerisindeki bir **if** bloğunda değer döndürmeden fonksiyonu sonlandırmak gerekiyorsa, tek başına "**return**" komutu kullanılmalıdır (Sağ tarafına herhangi bir şey yazılmamalıdır.). Örneğin, kendisine verilen iki sayıdan birinciyi ikinciye bölerek sonucu döndüren, ikinci sayı 0 ise sonuç döndürmeden uyarı veren bir fonksiyon aşağıdaki gibidir:

```
def TamsayiBolme(bolunen, bolen):
    sonuc = 0
    if bolen != 0:
        sonuc = bolunen/bolen
    else:
        print 'Bölen sayı sıfır olamaz.'
        return
    return sonuc
```

Çalıştırıldığında görülecek sonuç:

```
>>> TamsayiBolme(7,3)
2
>>> TamsayiBolme(8,0)
Bölen sayı sıfır olamaz.
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız.

Alıştırma – 3

Görev

Matematikte karesi negatif olan sayılara "**karmaşık sayılar**" denir. Karmaşık sayılar, $z = a + bi$ şeklinde ifade edilirler ($i = \sqrt{-1}$). Karmaşık sayıların mutlak değerleri ise **a** (karmaşık sayının gerçel kısmı) ve **b** (karmaşık sayının sanal kısmının katsayısı) değerleri kullanılarak $|z| = \sqrt{a^2 + b^2}$ şeklinde hesaplanır. Örneğin, $z_1 = 5 - 12\,i$ karmaşık sayısının mutlak değeri $|z_1| = \sqrt{5^2 + (-12)^2} = 13$ olarak bulunur.

Bu alıştırmada sizden, "**MutlakGenel.py**" isimli betik dosyasına "**MutlakGenel**" isminde argüman almayıp değer döndürmeyen ve gerek karmaşık sayıların, gerek tamsayıların, gerekse ondalıklı sayıların mutlak değerini hesaplayan bir fonksiyon yazmanız beklenmektedir.

Fonksiyon, kullanıcıya "Karmaşık sayılar için 1, tamsayılar için 2, ondalıklı sayılar için 3, çıkmak için 4 giriniz:" mesajını vermeli, kullanıcı **1** girerse karmaşık sayının gerçel kısmını (**a**) ve sanal kısmının katsayısını (**b**) alarak sonucu ekrana yazmalı; **2** girerse tamsayıyı, **3** girerse de ondalıklı sayıyı isteyerek bunların mutlak değerini ekrana yazmalıdır. Kullanıcı 4 girmediği sürece program çalışmalıdır. Örnek ekran görüntüsü aşağıdaki gibidir:

```
>>> MutlakGenel()

Karmaşık sayılar için 1, tamsayılar için 2, ondalıklı
sayılar için 3, çıkmak için 4 giriniz:
1
Gerçel kısmı giriniz                    :3
Sanal kısmı (i' nin katsayısını) giriniz :4

Kompleks sayının mutlak değeri : 5.0

Karmaşık sayılar için 1, tamsayılar için 2, ondalıklı
sayılar için 3, çıkmak için 4 giriniz:
2
Tamsayıyı giriniz:-4

Tamsayının mutlak değeri : 4

Karmaşık sayılar için 1, tamsayılar için 2, ondalıklı
sayılar için 3, çıkmak için 4 giriniz:
3
Ondalıklı sayıyı giriniz:-8.92

Ondalıklı sayının mutlak değeri : 8.92

Karmaşık sayılar için 1, tamsayılar için 2, ondalıklı
sayılar için 3, çıkmak için 4 giriniz:
4
>>>
```

İpucu

"**raw_input**" komutu ile kullanıcıdan aldığımız değerleri bir tamsayıya ya da ondalıklı sayıya çevirmemiz gerekebilir. Tamsayıya çevirmek için "**int()**", ondalıklı sayıya çevirmek içinse "**float()**" fonksiyonlarını aşağıdaki gibi kullanabiliriz:

```
>>> tamsayi = int(raw_input('Bir tamsayı giriniz:'))
Bir tamsayı giriniz:5
>>> type(tamsayi)
<type 'int'>
>>> ondaliklisayi = float(raw_input('Bir ondalıklı sayı
giriniz:'))
Bir ondalıklı sayı giriniz:7.43
>>> type(ondaliklisayi)
<type 'float'>
>>>
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız.

LABORATUVAR ÇALIŞMASI – 4

Bu Çalışmanın Amacı

Bu çalışmadaki amacımız, ardışık işlemler ve karakter dizileri konularındaki bilgilerimizi tekrarlamak ve uygulamalar yapmaktır.

Ardışık İşlemler

- Bir değişkene **birden çok kez** değer atandığında o değişkenin değeri, en son atanan değere eşittir.

```
>>> saat = '19.25'
>>> print saat
19.25
>>> saat = '19.27'
>>> print saat
19.27
```

- Bir **b** değişkenine bir **a** değişkenini atadığımızda artık **b** değişkeninin değeri de **a**' nın değeriyle aynı olacaktır. Ancak daha sonra **a**' nın değerini değiştirirsek bu **b**' nin değerinde herhangi bir değişikliğe sebep olmayacaktır.

```
>>> a = 8
>>> print a
8
>>> b = a
>>> print a
8
>>> print b
8
>>> a = -1
>>> print a
-1
>>> print b
8
```

- Eğer belli bir değere sahip bir değişkenin sağ tarafına "=" işareti koyup, onun da sağında bu değişkenin de dahil olduğu bir işlem yaparsak bu, "**Eşitliğin sağ tarafındaki işlemi yap, tamamla ve en sonunda bulduğun sonucu eşitliğin sol tarafındaki değişkenin içerisine at.**" anlamına gelmektedir. Örnek:

```
>>> x = 8
>>> print x
8
>>> x = x**2 + x/2
>>> print x
68
```

Break Komutu

Herhangi bir **for** ya da **while** döngüsünden çıkmak için kullanılır. Aşağıdaki betik dosyasını inceleyelim:

```
while True:
    komut = raw_input('Siz TAMAM yazana kadar bu program
sonlanmayacak. Birşeyler yazınız: ')
    if komut == 'TAMAM':
        break
    print '\nYAZDIĞINIZ ŞEY :', komut, '\n'
print '\nPROGRAM SONLANDI!'
```

Bu betik dosyasının örnek kullanımı ise şu şekildedir:

```
>>>
Siz TAMAM yazana kadar bu program sonlanmayacak. Birşeyler
yazınız: Merhaba

YAZDIĞINIZ ŞEY : Merhaba

Siz TAMAM yazana kadar bu program sonlanmayacak. Birşeyler
yazınız: 43210

YAZDIĞINIZ ŞEY : 43210

Siz TAMAM yazana kadar bu program sonlanmayacak. Birşeyler
yazınız: Tamam

YAZDIĞINIZ ŞEY : Tamam

Siz TAMAM yazana kadar bu program sonlanmayacak. Birşeyler
yazınız: TAMAM

PROGRAM SONLANDI!
>>>
```

Görüldüğü gibi **break** komutu, bizim istediğimiz herhangi bir şartın sağlanması durumunda anında döngü içerisinden çıkmamızı sağlamaktadır.

Karakter Dizileri

Bir veri türü olan karakter dizileri (**string**), bir veya birden fazla karakterin (boşluk karakteri de dâhil) arka arkaya gelmesinden oluşurlar ve değişken tanımlama esnasında tek tırnaklar ya da çift tırnaklar arasında ifade edilirler. Karakter dizileri, içerdikleri karakter adedi kadar uzunluk değerine sahiptirler. Bunların haricinde, içerisinde hiçbir karakter bulunmayan, sıfır uzunluk değerine sahip olan karakter dizileri de vardır. Eğer bir değişkene atama yaparken, eşitliğin sağındaki değeri tırnak içerisinde yazarsak Python yorumlayıcısı bu değişkenin türünü **karakter dizisi** olarak belirleyecektir.

```
>>> marka = "VOLVO"
>>> model = 's80'
>>> yil = "2005"
>>> ipotek = ''
>>> type(marka)
<type 'str '>
>>> type(model)
<type 'str '>
>>> type(yil)
<type 'str '>
>>> type(ipotek)
<type 'str '>
```

Yukarıda yer alan **ipotek** değişkeni, sıfır birim uzunluğunda bir karakter dizisidir. Karakter dizilerinin istediğimiz elemanına (herhangi bir karakterine) istediğimiz zaman, aşağıdaki gibi erişebiliriz:

```
>>> dil = 'PYTHON '
>>> print dil[0]
P
>>> print dil[3]
H
>>> print dil[-1]
N
>>> print dil[0.71]
***Hata Mesajı***
>>> print dil[19]
***Hata Mesajı***
```

Karakter dizisi içeren değişkenin herhangi bir sıradaki karakterine ulaşmak için gireceğimiz indeks değeri **tamsayı** olmalıdır ve mutlak değeri, değişkenin uzunluğunun **altında** olmalıdır. Negatif değer girildiğinde ise indeksleme **sağdan sola** doğru yapılır.

Bir karakter dizisinin uzunluğunu öğrenmek için "**len**" fonksiyonunu kullanabiliriz:

```
>>> dil = 'PYTHON'
>>> uzunluk = len(dil)
>>> uzunluk
6
>>> type(uzunluk)
<type 'int'>
```

len fonksiyonu, argüman olarak aldığı karakter dizisinin uzunluğunu **tamsayı** olarak döndürmektedir. Unutulmamalıdır ki Python' da indeksleme 0' dan başlar. Yani "PYTHON" sözcüğünün ilk harfi olan "P" harfi, bu sözcüğün birinci değil **sıfırıncı** harfi olarak kabul edilir. Bu nedenle, programlama yaparken karakter dizisinin ilk karakterinin indeksinin '**0**', son karakterinin indeksinin ise '**uzunluk - 1**' olduğu unutulmamalıdır:

```
>>> dil = 'PYTHON'
>>> uzunluk = len(dil)
>>> print dil[uzunluk-1]
6
>>> print dil[uzunluk]
***Hata Mesajı***
```

Karakter dizileri üzerinde karakter karakter işlem yapmak için **while** ve **for** komutlarından yararlanabiliriz. Bir karakter dizisinin her bir karakterini alt alta yazdırmak için aşağıdaki örnek betik dosyasını kullanabiliriz (solda). F5 tuşuna bastığımızda ekranda göreceğimiz çıktı ise sağdaki gibi olacaktır:

```
meyve = 'incir'
indeks = 0
while indeks < len(meyve):
    karakter = meyve[indeks]
    print karakter
    indeks = indeks + 1
```

```
i
n
c
i
r
```

Aynı işlemi, "**for** ***[karakter]*** **in** ***[karakter dizisi]*** " kalıbı ile de yapabiliriz:

```
meyve = 'incir'
for karakter in meyve:
    print karakter
```

```
i
n
c
i
r
```

Bu kalıp, karakter dizisini adım adım gezerek her bir değeri **karakter** değişkenine atacaktır.

Aynı kalıpla gerçekleştirebileceğimiz diğer bir örnek:

```
basharfler = 'BÇKMSTYZ'
govde = 'at'

for karakter in meyve:
    print karakter + govde
```

```
Bat
Çat
Kat
Mat
Sat
Tat
Yat
Zat
```

Karakter dizisini dilimlere ayırmak da mümkündür. Sonuç olarak **karakter dizisi** döndüren dilimleme komutlarının örnek açıklaması şu şekildedir:

sozcuk[: 3] → Sözcüğün **3.** karakterine kadar (3. karakter hariç) olan tüm karakterler (**0**, **1** ve **2** numaralı karakterler)

sozcuk[3 :] → Sözcüğün **3.** karakterinden sonraki (3. karakter de dahil) tüm karakterler (**3**, **4**, **5**, …, **[uzunluk - 1]**)

sozcuk[2 : 5] → Sözcüğün **2.**, **3.** ve **4.** karakterleri

sozcuk[:] → Sözcüğün tamamı

sozcuk[4 : 4] → *Sonuçta boş bir karakter dizisi dönecektir.*

Örnek kullanımı inceleyelim:

```
>>> deniz = 'Karadeniz'
>>> deniz[: 4]
'Kara'
>>> deniz[4 :]
'deniz'
>>> deniz[3 : 7]
'aden'
>>> deniz[:]
'Karadeiz'
>>> deniz[5 : 5]
''
>>> hece = deniz[2 : 4]
>>> hece
'ra'
>>> type(hece)
<type 'str'>
```

Herhangi bir karakter dizisinin herhangi bir karakterini alarak eğer küçük harfli ise büyük harfe dönüştürmek için "**upper()**" fonksiyonu kullanılır. **upper** fonksiyonu orijinal karakter dizisinde herhangi bir değişiklik yapmaz, sadece büyük harfe çevirdiği kısmı yeni bir karakter dizisi olarak döndürür:

```
>>> model = 'bmw730d'
>>> basharf = model[0].upper()
>>> basharf
'B'
>>> type(basharf)
<type 'str'>
>>> model[3 :].upper()
'730D'
>>> model.upper()
'BMW730D'
```

Bir karakter dizisinin içerisinde başka bir karakter dizisinin bulunup – bulunmadığını "**find()**" fonksiyonu ile öğrenebiliriz. Eğer bulunursa kaçıncı indeksten itibaren yer aldığına dair başlangıç indeksinin numarasını, bulunamaz ise de **-1** döndürür. **find** fonksiyonuna, karakter dizisinin hangi indeksleri aralığında arama yapacağını da parametre olarak verebiliriz. Aşağıdaki örneği inceleyelim:

```
>>> sozcuk = 'istihza'
>>> sozcuk.find('t')
2
>>> sozcuk.find('ihz')
3
>>> sozcuk.find('ihs')
-1
>>> sozcuk.find('i')
0
>>> sozcuk.find('i', 1)
3
>>> sozcuk.find('i', 1, 3)
-1
>>> sozcuk.find('i', 1, 4)
3
```

Dördüncü işlemde "**i**" karakteri, "**istihza**" sözcüğünün 2. karakterinden (Python' a göre 1. karakterinden) itibaren aranmış olduğu için aramaya "s" harfinden başlanmış ve ikinci "i" harfine rastlanarak indeks değeri **3** olarak hesaplanmıştır. Beşinci işlemde ise "**i**" karakteri, dizinin sadece 2. ve 3. karakterleri içerisinde aranmış ve bulunamadığı için **-1** elde edilmiştir.

Bir karakter dizisinin içerisinde başka bir karakter dizisinin bulunup – bulunmadığını öğrenmek için "**in**" operatörü kullanılır. Sonuç olarak "**bool**" türünde olan "**True** (doğru)", ya da "**False** (yanlış)" değerlerinden birini döndürür:

```
>>> sozcuk = 'istihza'
>>> 'a' in sozcuk
True
>>> 'p' in sozcuk
False
>>> 'ihz' in sozcuk
True
>>> 'ihs' in sozcuk
False
```

Karakter dizileri arasında **eşitlik**, **büyük/küçük/büyük eşit/küçük eşit olma** durumlarını kontrol etmek mümkündür. İki sözcükten **sözlük sırasına göre sonra gelen**, diğerinden daha **büyüktür**:

```
>>> sozcuk1 = 'yalan'
>>> sozcuk2 = 'yanlis'
>>> sozcuk3 = 'yanilma'
>>> sozcuk4 = 'yanilis'
>>> sozcuk5 = 'yanlis'
>>> sozcuk1 > sozcuk2
False
>>> sozcuk1 == sozcuk2
False
>>> sozcuk1 < sozcuk2
True
>>> sozcuk2 == sozcuk5 > sozcuk3 > sozcuk4 > sozcuk1
True
>>> sozcuk2 == sozcuk5 > sozcuk3 > sozcuk4 < sozcuk1
False
>>> sozcuk3 > sozcuk4 == sozcuk1
False
```

NOT: Bu sıralama ASCII kullanılarak yapıldığı ve Türkçe karakterlerin ASCII değerleri alfabetik sıralamaya uygun olmadığı için karşılaştırma işlemlerinde Türkçe karakter kullanmaktan uzak durmalıyız.

Alıştırmalar

Alıştırma – 1

Görev

Çarpma işlemi yapmadan, sadece toplama işlemi yaparak **0**' dan bir pozitif **n** tamsayısına kadar olan tamsayıların toplamını (1 + … + n) hesaplayan bir fonksiyon tanımlayınız. Fonksiyon, **n** tamsayısını parametre olarak almalı ve pozitif bir tamsayı girilmediğinde "Lütfen pozitif bir tamsayı giriniz." uyarı mesajını yazdırmalıdır. Fonksiyonunuza "ArdisikToplam" ismini vererek "ArdisikToplam.py" isimli bir betik dosyasına kaydediniz. Fonksiyonun çalıştırılmasına ait örnek ekran görüntüsü aşağıdadır:

```
>>> ArdisikToplam(7)
28
>>> ArdisikToplam(0)
Lütfen pozitif bir tamsayı giriniz.
>>> ArdisikToplam(-7)
Lütfen pozitif bir tamsayı giriniz.
>>> ArdisikToplam(7.1)
Lütfen pozitif bir tamsayı giriniz.
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 2

Görev

Bir pozitif sayıyı, sayının kendisi defa sürekli (bölme işleminin sonucunu da ikiye bölerek devam eden bir biçimde) ikiye bölen ve sonuç olarak da tüm bölümlerin toplamını döndüren bir fonksiyon yazınız. Örneğin, fonksiyona **3** değeri verildiğinde:

3 / 2 = **1.5**

1.5 / 2 = **0.75**

0.75 / 2 = **0.375** (Tamsayı **3** olduğu için 3. basamakta duruldu.)

1.5 + **0.75** + **0.375** = **2.625**

Ancak her bir basamakta hesaplanan **bölüm** değeri **0.1' e eşit ya da 0.1' den küçük olursa** sürekli bölme işlemi o basamakta sona erdirilerek 0.1' den büyük olan bölümler toplanarak döndürülmelidir. Fonksiyona 7 verildiğinde:

7 / 2 = **3.5**

3.5 / 2 = **1.75**

1.75 / 2 = **0.875**

0.875 / 2 = **0.4375**

0.4375 / 2 = **0.21875**

0.21875 / 2 = **0.109375**

0.109375 / 2 = **0.0546875**

(0.0546875 sayısı 0.1' den küçük olduğu için hesaba katılmaz. Böylesi bir durumda sayı 7' den büyük olsaydı ve bölme işleminin devam etmesi gerekseydi bile sürekli bölme işlemi bu noktada kesilmelidir.**)**

3.5 + **1.75** + **0.875** + **0.4375** + **0.21875** + **0.109375** = **6.890625**

Programa pozitif tamsayı dışında bir sayı girildiğinde ekrana "Lütfen pozitif bir tamsayı giriniz." mesajı yazılmalıdır.

Fonksiyona “YarilarToplami” ismini veriniz ve “YarilarToplami.py” isimli bir betik dosyasına kaydediniz. Fonksiyonun çalıştırılmasına dair örnek gösterim aşağıdadır:

```
>>> YarilarToplami(2)
1.5
>>> YarilarToplami(3)
2.625
>>> YarilarToplami(5)
4.84375
>>> YarilarToplami(7)
6.890625
>>> YarilarToplami(0)
Lütfen pozitif bir tamsayı giriniz.
>>> YarilarToplami(-4)
Lütfen pozitif bir tamsayı giriniz.
>>> YarilarToplami(6.8)
Lütfen pozitif bir tamsayı giriniz.
```

İpucu

“**break**” komutunu inceleyiniz.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 3

Görev

Parametre olarak **Türkçe karakter** (Ç, ç, Ğ, ğ, ı, İ, Ö, ö, Ş, ş, Ü, ü) **içermeyen** bir karakter dizisi alan ve sonuçta o karakter dizisini tersten yazılmış halini döndüren bir fonksiyon yazınız. Fonksiyonunuza "SondanBasaYaz" ismini veriniz ve "SondanBasaYaz.py" isimli bir betik dosyasına kaydediniz. Fonksiyonunuza parametre olarak karakter dizisi dışında bir değer girildiğinde ekrana "Lütfen parametre olarak karakter dizisi giriniz." yazısı yazılmalıdır. Fonksiyonun kullanımına dair örnek aşağıdadır:

```
>>> SondanBasaYaz('Ondokuz Mayis Universitesi')
'isetisrevinU siyaM zukodnO'
>>> SondanBasaYaz('')
''
>>> SondanBasaYaz(45)
Lütfen parametre olarak karakter dizisi giriniz.
>>> SondanBasaYaz(9.87)
Lütfen parametre olarak karakter dizisi giriniz.
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

LABORATUVAR ÇALIŞMASI – 5

Bu Çalışmanın Amacı

Bu çalışmadaki amacımız, metin dosyalarından yazı okuyarak okunan bu yazılar üzerinde işlem yapma ve liste kullanımı konularında öğrendiklerimizi pekiştirmektir.

Dosyadan Veri Okuma

Python' da bir dosyayı okuyabilmek için öncelikle açmak gerekir. Dosyayı açmak içinse "**open**" fonksiyonu kullanılır. **open** fonksiyonu parametre olarak dosya adresini (dosya ismine kadar, '*C:\...\...\...\dosya.txt*' gibi) alır, sonuç olaraksa bir dosya nesnesi döndürür.

Dosyadan okuma işlemleri, işte bu dosya nesnesi üzerinden yapılır. Örneğin, "sozcuk.txt" isminde bir metin dosyamız olsun ve içeriği aşağıdaki gibi olsun:

```
aba
abajur
abanmak
abanoz
abartmak
abide
abiye
abone
```

Bu dosyanın çalışma dizininde (betik dosyalarının olduğu dizinde) bulunduğunu düşünelim ve bu dosyadan satır satır okuma yaparak okunan kelimeleri alt alta ekrana yazdıran bir program hazırlayalım (Python Shell ekranında). **open** fonksiyonunun döndüreceği değeri içerisine atacağımız dosya nesnesine, "dosya girdisi (file input)" anlamında "**fin**" ismini verelim (Genelde bu isim verilir.):

```
>>> fin = open('sozcuk.txt')
>>> for satir in fin:
        print satir


aba

abajur

...

abone
```

Kodu bu şekilde yazdığımızda sözcüklerin ekrana birer satır arayla yazıldığını görürüz. Bunun nedeni, hem **print** komutunun her defasında yeni bir satıra yazması, hem de dosyadan

aba\r\n
abajur\r\n
abanmak\r\n
abanoz\r\n
abartmak\r\n
abide\r\n
abiye\r\n
abone\r\n

okunan her bir sözcüğün sonunda, "**Bir alt satıra geç.**" anlamına gelen "\r\n" karakterlerinin bulunmasıdır. Metin dosyasında biz bu "**\r\n**" karakterlerini yazılı olarak görmeyiz, çünkü bu karakterlerin görevi zaten **metnin bir alt satırdan devam edeceğini göstermek**tir. Eğer "sozcuk.txt" dosyasından bir şekilde bu "**\r\n**" karakterlerini yok edersek elde edeceğimiz yeni metin dosyasını açtığımızda göreceğimiz şey şu şekilde olacaktır:

abaabajurabanmakabanozabartmakabideabiyeabone

Ancak "**readline**" komutu ile okuma yaptıktan sonra "**\r\n**" karakterlerini atmak da mümkündür. Bunun içinse "**strip**" fonksiyonu kullanılır. Örnek kullanım aşağıdaki gibidir:

```
>>> fin = open('sozcuk.txt')
>>> for satir in fin:
        ArindirilmisSatir = satir.strip()
        print ArindirilmisSatir


aba
abajur
abanmak
abanoz
abartmak
abide
abiye
abone
```

"ArindirilmisSatir" değişkeni içerisinde "**\r\n**" karakterlerinden arındırılmış sözcükler bulunduğu için ve **print** komutu her defasında yeni bir satırdan yazmaya başladığı için kelimeler arasında boş satırlar yer almayacaktır.

Listeler

Python' da aynı veya farklı türde bir ya da birden çok değişkeni içerisinde bulunduran yapılara "liste" adı verilir. Listeler köşeli parantezler ile ("**[**", "**]**") ifade edilirler ve birden çok elemana sahip olan listelerde köşeli parantezler arasına yazılan elemanlar birbirinden virgülle ayrılır. Bir listenin elemanı bir ya da birden çok elemana sahip olan bir ya da birden fazla sayıda liste olabilir. Örnekler:

```
>>> OndanKucukAsalSayilar = [2, 3, 5, 7]
>>> OndanKucukAsalSayilar
[2, 3, 5, 7]
>>> OndanKucukAsalSayilar[3]
7
>>> OndanKucukAsalSayilar[0]
2
>>> OndanKucukAsalSayilar[4]
***Hata Mesajı (olmayan bir indekse erişimden dolayı)***
>>> SehirBilgileri = ['Samsun', 55, "Karadeniz"]
>>> SehirBilgileri
['Samsun', 55, "Karadeniz"]
>>> SehirBilgileri[0]
'Samsun'
>>> SehirBilgileri[1]
55
>>> type(SehirBilgileri[0])
<type 'str'>
>>> type(SehirBilgileri[1])
<type 'int'>
>>> Bolge = ["Karadeniz", ['Samsun', 55], ['Zonguldak', 67]]
>>> Bolge
["Karadeniz", ['Samsun', 55], ['Zonguldak', 67]]
>>> Bolge[0]
'Karadeniz'
>>> Bolge[1]
['Samsun', 55]
>>> Bolge[2][0]
'Zonguldak'
>>> Bolge[2][1]
67
>>> type(Bolge[2][1])
<type 'int'>
```

Örnekte de gördüğümüz gibi listelerin elemanlarına, hatta liste içerisinde yer alan listelerin elemanlarına, indekslerini belirterek (köşeli parantezler içerisinde) ulaşmamız mümkündür.

Herhangi bir listenin herhangi bir elemanını sonradan değiştirmek mümkündür:

```
>>> Sehir = ['Tokat', 60, 'Marmara']
>>> Sehir
['Tokat', 60, 'Marmara']
>>> Sehir[2]
'Marmara'
>>> Sehir[2] = 'Karadeniz'
>>> Sehir
['Tokat', 60, 'Karadeniz']
>>> Sehir[2]
'Karadeniz'
```

Herhangi bir değerin liste içerisinde yer alıp - almadığını kontrol etmek için "**in**" komutu kullanılır. Sonuç olarak, **bool** türünde **True** ya da **False** dönecektir. Örnek:

```
>>> Sehir = ['Tokat', 60, 'Karadeniz']
>>> 'Tokat' in Sehir
True
>>> 60 in Sehir
True
>>> 'Toka' in Sehir
False
>>> 6 in Sehir
False
>>> 600 in Sehir
False
```

"***for*** *[eleman]* **in** *[liste]*" kalıbı kullanılarak liste içerisinde gezilebilir:

```
>>> Sehir = ['Izmir', 35, ['Konak', 'Buca', 'Karsiyaka']]
>>> Sehir
['Izmir', 35, ['Konak', 'Buca', 'Karsiyaka']]
>>> for HerBirEleman in Sehir:
        Print HerBirEleman


Izmir
35
['Konak', 'Buca', 'Karsiyaka']
```

for döngüsü ile listeler üzerinde gezerek işlem yapılabilir:

```
>>> AsalSayilar = [2, 3, 5, 7]
>>> AsalSayilar
[2, 3, 5, 7]
>>> for i in range(0, len(AsalSayilar)):
        AsalSayilar[i] = AsalSayilar[i] * 2


>>> AsalSayilar
[4, 6, 10, 14]
```

Bir listenin içerisinde yer alan diğer listelerden her biri tek bir eleman olarak değerlendirilir:

```
>>> dizi1 = [1, 2, 3]
>>> dizi2 = [1, [2, 3, 4, 5], ['a', 'b', 'c', 6]]
>>> len(dizi1)
3
>>> len(dizi2)
3
```

"+" operatörü iki listeyi arka arkaya birleştirirken "*" operatörü de listeyi, kendisinin sağındaki tamsayı kadar tekrar ettirerek arka arkaya birleştirir:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 'k']
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5, 'k']
>>> d = b * 3
>>> print d
[4, 5, 'k', 4, 5, 'k', 4, 5, 'k']
```

Listeler, dilimlere ayrılabilirler:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1 : 3]
['b', 'c']
>>> t[:4]
['a', 'b', 'c', 'd']
>>> t[3:]
['d', 'e', 'f']
>>> t[:]
['a', 'b', 'c', 'd', 'e', 'f']
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1 : 3] = ['x', 'y']
>>> print t
['a', 'x', 'y', 'd', 'e', 'f']
```

"**append**" komutu, listenin en sonuna yeni bir eleman eklemede kullanılır:

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print t
['a', 'b', 'c', 'd']
```

"**extend**" komutu, bir listenin sonuna diğer bir listeyi eklemek için kullanılır:

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> print t1
['a', 'b', 'c', 'd', 'e']
>>> print t2
['d', 'e']
```

Listenin elemanlarını küçükten büyüğe doğru sıralamak için "**sort**" komutu kullanılır:

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> print t
['a', 'b', 'c', 'd', 'e']
>>> u = ['d', 4, 'c', -1, 7, 'e', 0, 'b', 1.7, 'a']
>>> u.sort()
>>> print u
[-1, 0, 1.7, 4, 7, 'a', 'b', 'c', 'd', 'e']
```

Listenin herhangi bir indeksindeki elemanı çıkartıp bir değişkene atmak için "**pop**" komutu kullanılır:

```
>>> t = ['a', 'b', 'c']
>>> print t[1]
b
>>> x = t.pop(1)
>>> print t
['a', 'c']
>>> print t[1]
c
>>> print x
b
```

Eğer listeden çıkarılmak istenen eleman artık kullanılmayacaksa "**del**" komutu ile silinebilir. Bu komut, dilimleme ile birlikte de kullanılabilir:

```
>>> t = ['a', 'b', 'c']
>>> del t[1]
>>> print t
['a', 'c']
>>> u = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del u[1 : 5]
>>> print u
['a', 'f']
```

Listeden silinecek elemanın indeksi bilinmiyor ama değeri biliniyorsa "**remove**" komutu kullanılabilir:

```
>>> t = ['a', 'b', 'c']
>>> t.remove('b')
>>> print t
['a', 'c']
```

Bir karakter dizisini bir listeye karakter karakter atmak için "**list**" komutu kullanılır:

```
>>> s = 'python'
>>> t = list(s)
>>> print t
['p', 'y', 't', 'h', 'o', 'n']
```

Boşluklar içeren bir karakter dizisinin boşluklarla ayrılmış olan her bir harf grubunu bir listenin indekslerine sırasıyla atmak için "**split**" komutu kullanılır:

```
>>> s = 'Python ile programlama yapmak'
>>> t = s.split()
>>> print t
['Python', 'ile', 'programlama', 'yapmak']
```

Yukarıdaki işlemde ayırıcı olarak boşluk karakterini (" ") kullanmıştık. Aynı içlem için başka karakterler kullanmak da mümkündür. Örnek kullanım ("**-**" karakteri için) şu şekildedir:

```
>>> s = 'Python2.4.2-Python2.6.4-Python3.1.1'
>>> delimiter = '-'
>>> s.split(delimiter)
['Python2.4.2', 'Python2.6.4', 'Python3.1.1']
```

split komutunun tersi göreve sahip olan komut ise "**join**" dir:

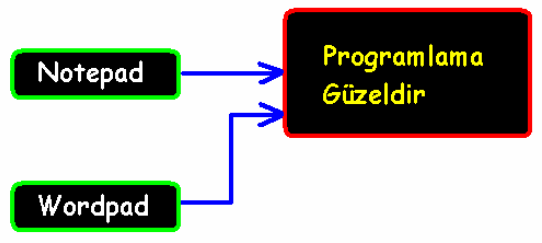
```
>>> t = ['Python2.4.2', 'Python2.6.4', 'Python3.1.1']
>>> delimiter = '+'
>>> delimiter.join(t)
'Python2.4.2+Python2.6.4+Python3.1.1'
```

Python' da **değişken**ler, bellekte belli bir adrese sahip olan **nesne**lerin adreslerini tutan **tutacaklardır**. Örneğin, Windows bilgisayarımızın masaüstünde bir **metin belgesi** oluşturup içerisine bir şeyler kaydedip kapattıktan sonra bunu hem **Notepad** metin editörü ile, hem de **Wordpad** metin editörü ile açabiliriz. Burada metin belgesini **nesne**, her iki metin editörünü de **iki ayrı tutacak** gibi düşünebiliriz. Belgeyi **Notepad** ile açıp bir değişiklik yapıp kaydedersek, **Wordpad** ile açtığımızda değişiklik yapılmış hali ile karşılaşırız. Oysa iki ayrı metin dosyası belirleyerek birisine "**N.txt**", diğerine ise "**W.txt**" isimlerini verip, **N.txt**' yi sadece **Notepad** ile, **W.txt**' yi ise sadece **Wordpad** ile kullanıyor olsaydık, her bir metin editörü ile yapılan değişiklik, sadece o editörle açtığımız belgelerde karşımıza çıkacaktı.

Yukarıda açıklanan durumun mantığını göz önünde bulundurarak aşağıdaki kodları inceleyelim:

```
>>> Notepad = 'Programlama guzeldir.'
>>> Wordpad = Notepad
>>> Notepad
'Programlama guzeldir.'
>>> Wordpad
'Programlama guzeldir.'
>>> Notepad is Wordpad
True
```

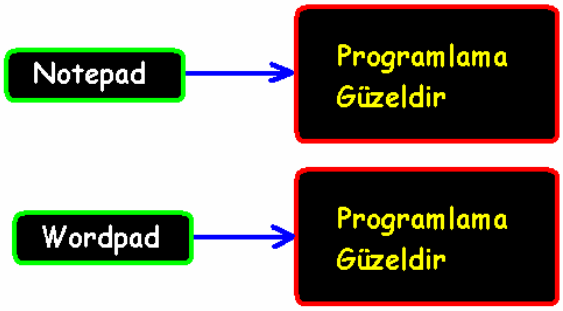
İki değişkenin (yani **tutacağın**) gösterdiği yerdeki nesnelerin **aynı nesne** olup - olmadığını öğrenmek için "**is**" komutu kullanılır.



Şimdi ise aşağıdaki kodun yukarıdakinden farklılığını irdeleyelim:

```
>>> Notepad = 'Programlama guzeldir.'
>>> Wordpad = 'Programlama guzeldir.'
>>> Notepad
'Programlama guzeldir.'
>>> Wordpad
'Programlama guzeldir.'
>>> Notepad is Wordpad
False
```

Burada ise iki farklı **tutacak** için **iki farklı nesne** bulunmaktadır. Tutacakların gösterdikleri yerlerdeki nesneler farklı farklı nesnelerdir (her ne kadar içerikleri aynı olsa da).



Aşağıdaki örnekte ise, nesnenin tutacaklar tarafından paylaşılması durumunda tutacaklardan biri ile nesnenin değiştirilmesi sonucunda değişikliğin diğer tutacağa da yansıdığı görülmektedir:

```
>>> a = [1, 2, 3]
>>> b = a
>>> b is a
True
>>> b[0] = 17
>>> print a
[17, 2, 3]
>>> print b
[17, 2, 3]
```

Listeler, tıpkı diğer değişkenler gibi (karakter dizisi, ondalıklı sayı vs.) fonksiyonlara **argüman** olarak verilebilirler. Aşağıda, kendisine argüman olarak verilen listenin ilk elemanını silen bir fonksiyon verilmiştir (betik dosyasında):

```
def IlkElemaniSil(liste):
    del liste[0]
```

Bu fonksiyonun Python Shell' de kullanımını inceleyelim:

```
>>> kent = ['a', 'n', 'k', 'a', 'r', 'a']
>>> IlkElemaniSil(kent)
>>> kent
['n', 'k', 'a', 'r', 'a']
```

Alıştırmalar

Alıştırma – 1

Görev

"Sozcuk.txt" dosyasındaki 5 karakterden daha uzun (minimum 6 karaktere sahip) olan sözcükleri ekrana yazdıracak komutları yazınız (Python Shell ekranında). Satır sonundaki "\r\n" karakterlerini yok etmeyi unutmayınız. Komutlar çalıştırıldığında ekranda görülecek olan çıktı aşağıdaki gibi olmalıdır:

```
abajur
abanmak
abanoz
abartmak
>>>
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 2

Görev

Bir metin dosyasından okuma yaparak içerisinde belli bir karakteri bulundurmayan satırları ekrana yazdıran bir fonksiyon yazınız. Fonksiyon, 1. parametre olarak okuma yapılacak olan dosyanın adını (**DosyaAdi**), ikinci parametre olarak ise satırlarda yer almasını istemediğimiz karakteri (**Karakter**) almalıdır ve sonuçları ekrana yazdırmalıdır (Yani değer olarak döndürmemelidir.). Fonksiyonunuza "karakter_filtre" ismini vererek "karakter_filtre.m" adlı bir betik dosyasına kaydediniz. Fonksiyonun örnek kullanımı aşağıdaki gibi olmalıdır:

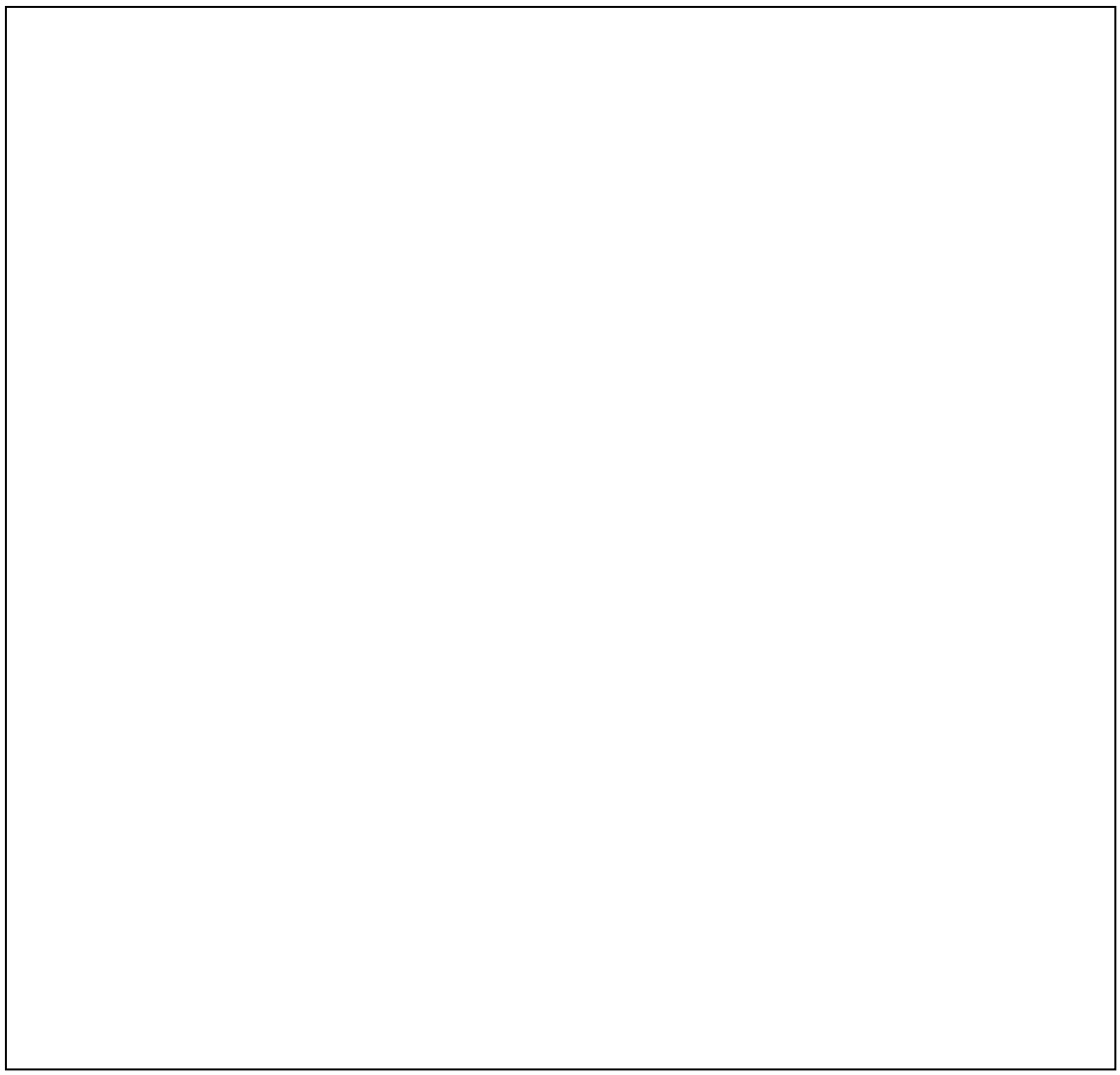
```
>>> karakter_filtre('Sozcuk.txt', 'n')
aba
abajur
abartmak
abide
abiye
>>> karakter_filtre('Sozcuk.txt', 'j')
aba
abanmak
abanoz
abartmak
abide
abiye
abone
>>> karakter_filtre('Sozcuk.txt', 'a')
>>> karakter_filtre('Sozcuk.txt', 'e')
aba
abajur
abanmak
abanoz
abartmak
```

İpucu

"Karakter Dizileri" bölümünü ve "**for** ***[karakter]*** **in** ***[karakter dizisi]***" kalıbını inceleyiniz.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:



Alıştırma – 3

Görev

Bir sözcüğün içerisindeki karakterlerin soldan sağa alfabetik sıraya göre dizilip - dizilmediğini kontrol eden bir fonksiyon yazınız. Fonksiyona "AlfabetikArtan" ismini veriniz ve "AlfabetikArtan.py" isimli betik dosyasına kaydediniz. Fonksiyon argüman olarak sözcük almalı, değer olaraksa **bool** türünde True (sözcüğün alfabetik olarak dizildiği durumlar için) ya da False (diğer durumlar için) döndürmelidir. Fonksiyona sadece küçük harflerden oluşan ve Türkçe harf içermeyen sözcükler girilmelidir. Örnek kullanım aşağıdadır:

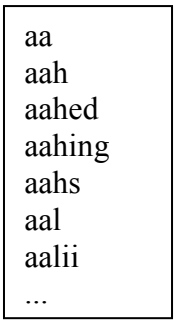
```
>>> AlfabetikArtan('adem')
True
>>> AlfabetikArtan('koop')
True
>>> AlfabetikArtan('kaan')
False
>>> AlfabetikArtan('emre')
False
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 4

Görev



Size tüm İngilizce kelimelerin alfabetik sıraya göre ve satır satır yer aldığı "words.txt" isimli bir metin dosyası verilecektir. Bu dosyanın başlangıç kısmının görünümü yandaki gibidir. Alıştırma - 1 ve Alıştırma - 3' teki gerçekleştirimlerinizi sentezleyerek, İngilizce' de alfabetik artan sırada olan tüm sözcükleri ekrana yazdıracak olan bir fonksiyon hazırlayınız.

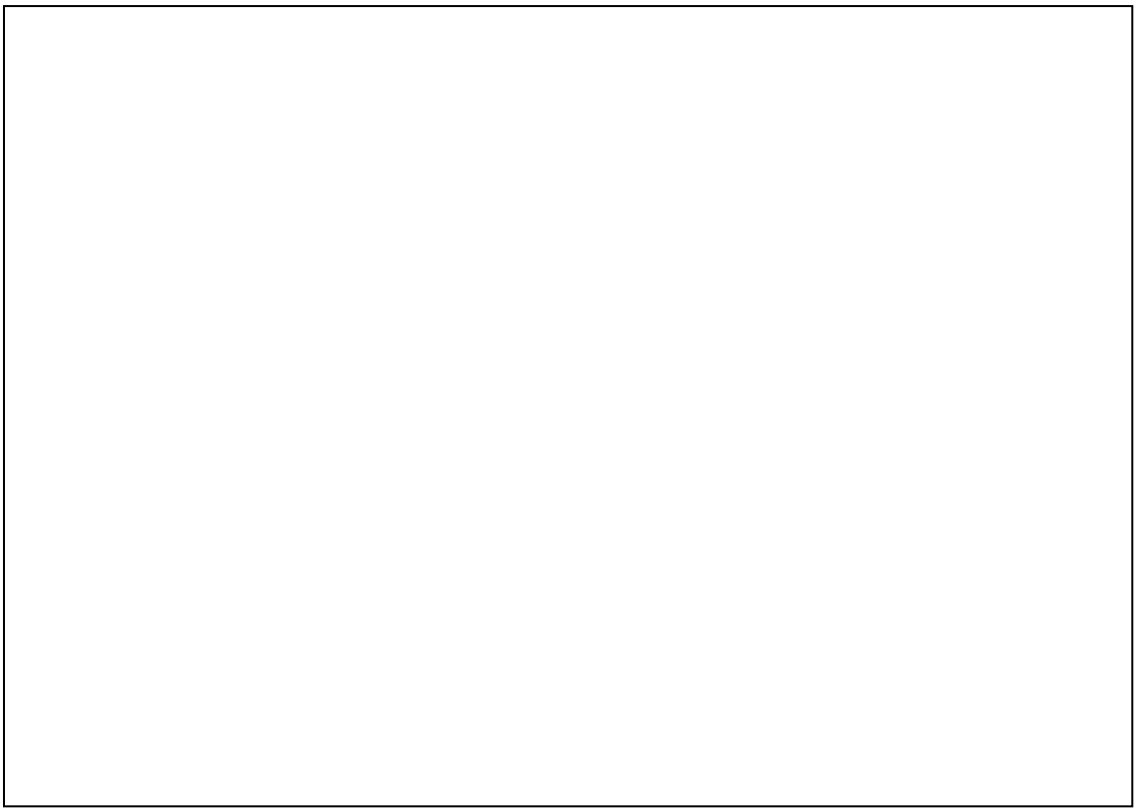
Fonksiyonunuza "IngilizceAlfArtan" ismini vererek "IngilizceAlfArtan.py" isimli betik dosyasına kaydediniz. Fonksiyon parametre almamalı ("words.txt" yi kendi içinde çağırmalı) ve değer döndürmemelidir (Değer döndürmek yerine ekrana yazdırmalıdır.). En son satırda ise alfabetik olarak artan sözcük sayısını ekrana yazmalıdır.Fonksiyon çalıştırıldığında elde edilecek olan ekran görüntüsü aşağıdaki gibi olmalıdır.

```
>>> IngilizceAlfArtan()
aa
aah
aahs
aal
aals
...
...
...
psst
sty
tux

Alfabetik artan sozcuk sayisi: 596
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:



Alıştırma – 5

Görev

Argüman olarak bir karakter dizisi alan ve sonuç olarak ekrana, bu dizinin elemanlarından kaçının karakter dizisi, kaçının tamsayı, kaçının ondalıklı sayı, kaçının "bool", kaçının liste olduğunu yazan, değer döndürmeyen bir fonksiyon yazınız. Fonksiyonunuza "ListeAnalizi" ismini veriniz ve "ListeAnalizi.py" isimli bir betik dosyasına kaydediniz. Fonksiyonun, argüman olarak verilen listenin içinde eleman olarak bulunan diğer listelerin içerikleriyle ilgilenmesine lüzum yoktur. Fonksiyonun çalıştırılmasına dair örnek ekran görüntüsü aşağıdadır:

```
>>> Su = ['H2O', 18, 0, 100.0, 'Bilesik', "Kovalent",
['Hidrojen', 2], ['Oksijen', 1], 3, True]
>>> ListeAnalizi(Su)
Karakter dizisi sayısı : 3
Tamsayı sayısı         : 3
Ondalıklı sayı sayısı  : 1
Boolean sayısı         : 1
Liste sayısı           : 2
>>> ListeAnalizi( [1, '1.1', [1.1, 1.1], [False, 7, True], -1] )
Karakter dizisi sayısı : 1
Tamsayı sayısı         : 2
Ondalıklı sayı sayısı  : 0
Boolean sayısı         : 0
Liste sayısı           : 2
```

İpucu

"**lis**" isimli bir dizinin eleman sayısını bulmak için "**len(lis)**" komutu kullanılabilir.

Bir **a** değişkeninin değerinin tamsayı olup - olmadığını kontrol etmek için "**type(a)==type(7)**" ifadesi kullanılabilir.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 6

Görev

Elimizde "-" işaretleri ile ayrılmış, "6-19-4-60-22-7-4" şeklinde, pozitif tamsayılardan oluşan karakter dizileri var. **a**, bu dizideki tamsayıların ortalaması (tamsayıların toplamının tamsayı adedine bölümü [tamsayı bölme]) olsun. **b** ise, bu dizi küçükten büyüğe doğru sıralandığında, eğer tek sayıda eleman varsa en ortadaki (5 tane tamsayı varsa baştan 3.) sayı, çift sayıda eleman varsa ortadaki iki sayının ortalaması (6 tane tamsayı varsa 3. ve 4. sayının toplamının yarısı [tamsayı bölme ile]) olsun. **a * b** ise bu dizinin **anahtarı** olsun.

Argüman olarak yukarıdaki gibi bir tamsayı dizisi alan, sonuç olarak ise dizinin anahtarını döndüren bir fonksiyon hazırlayınız. Fonksiyonunuza "DiziAnahtari" ismini vererek "DiziAnahtari.py" isimli bir betik dosyasına kaydediniz.

NOT: Fonksiyonunuza yukarıdaki formatta ve en az 3 tane pozitif tamsayıdan oluşan karakter dizilerinin **doğru** bir biçimde girileceği garanti edilmektedir. Aksi durumlar için hata kontrolü yapmanıza **lüzum yoktur**.

Örnek olarak **'3-10-7-5'** dizisinde çift sayıda eleman vardır. Bunları küçükten büyüğe doğru sıraladığımızda **'3-5-7-10'** elde ederiz. **a=(3 + 5 + 7 + 10) / 4 = 25 / 4 = 6**,
b = (5 + 7) / 2 = 12 / 2 = 6 olacağından anahtar değeri, **6 * 6 = 36** bulunur.

'3-10-7' dizisinde ise tek sayıda eleman vardır. Bunları küçükten büyüğe doğru sıraladığımızda **'3-7-10'** elde ederiz. **a=(3 + 7 + 10) / 3 = 20 / 3 = 6**, **b = 7** olacağından anahtar değeri, **6 * 7 = 42** olarak bulunur. Örnek kullanım, aşağıdadır:

```
>>> DiziAnahtari('3-5-7-10')
36
>>> DiziAnahtari('3-10-7-5')
36
>>> DiziAnahtari('3-10-7')
42
>>> DiziAnahtari('7-3-10')
42
```

İpucu

'5' karakterini tamsayıya dönüştürmek için **int('5')** yazılabilir. **a='5'** için ise **int(a)** yazılmalıdır. **int**, bize dönüşümün sonucunu tamsayı olarak verir.

Bir **a** tamsayısının tek mi çift mi olduğunu öğrenmek için **(a/2)** ile **(a/2.0)** ifadelerinin eşit olup - olmadığına bakılır. Bu ikisi eşit ise sayı çifttir, değilse sayı tektir.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

LABORATUVAR ÇALIŞMASI – 6

Bu Çalışmanın Amacı

Bu çalışmadaki amacımız, Python' da yer alan sözlük (dictionary) ile 'tuple' veri türleri ve global değişkenler hakkında öğrendiklerimizi pekiştirmektir.

Sözlük

Sözlükleri, **indeks değerleri tamsayılardan oluşabileceği gibi, karakter dizilerinden de oluşabilen listeler** olarak tanımlayabiliriz. Sözlüklerde de listelerde olduğu gibi, herhangi bir indekste herhangi bir türde (karakter dizisi, tamsayı, ondalıklı sayı vs.) veri tutulabilir.

Bir sözlüğün tanımlanmasını ve kullanılmasını gösteren örnek aşağıdadır:

```
>>> TurIng = dict()
>>> TurIng['bir'] = 'one'
>>> TurIng['iki'] = 'two'
>>> TurIng['uc'] = 'three'
>>> TurIng['bes'] = 'five'
>>> TurIng['iki']
'two'
>>> type(TurIng['uc'])
<type 'str'>
>>> TurIng['dort']
***Hata Mesajı (olmayan bir indekse erişimden dolayı)***
>>> TurIng
{'bes': 'five', 'iki': 'two', 'bir': 'one', 'uc': 'three'}
>>> type(TurIng)
<type 'dict'>
>>> len(TurIng)
5
```

Sözlüğü tek bir satırda tanımlamak da mümkündür:

```
>>> TurIng = dict()
>>> TurIng = {'bir': 'one', 'iki': 'two', 'uc': 'three',
                                          'bes': 'five'}
>>> TurIng['uc']
'three'
```

Sözlükte yer alan indislerin isimlerini (indisler içerisine atanan değerleri değil) kontrol etmek için **in** komutu kullanılabilir. Aranan indeks isminin sözlükte bulunması durumunda **True**, diğer durumda ise **False** döndürür:

```
>>> TurIng = dict()
>>> TurIng = {'bir': 'one', 'iki': 'two', 'uc': 'three'}
>>> 'iki' in TurIng
True
>>> 'two' in TurIng
False
>>> 'ik' in TurIng
False
```

Eğer değerlerin bulunup bulunmadığını kontrol etmek istiyorsak, "**values**" isimli fonksiyondan şu şekilde yararlanabiliriz:

```
>>> TurIng = dict()
>>> TurIng = {'bir': 'one', 'iki': 'two', 'uc': 'three'}
>>> degerler = TurIng.values()
>>> 'two' in degerler
True
>>> 'iki' in degerler
False
>>> 'tw' in degerler
False
```

Sözlüklerde indeks değerlerleri tamsayılardan da oluşabilir:

```
>>> Kare = dict()
>>> Kare = {0 : 0, 1 : 1, 2 : 4, 3 : 9, 4 : 16, 5 : 25}
>>> Kare[0]
0
>>> Kare['0']
***Hata Mesajı***
>>> Kare[4]
16
```

Global Değişkenler

Python' da bir fonksiyonun dışında (fonksiyon tanımlaması ile aynı girinti ['indent'] değerine sahip olacak biçimde) tanımlanan bir değişken, fonksiyonun dışında kullanılabilir, değeri değiştirilebilir. Fonksiyonun içinden değeri görülebilse de **direkt atama yapılarak değiştirilemez**. Aşağıdaki betik dosyasında yer alan örnek kodları inceleyelim:

```
Bildirim = True
def OrnekFonk1():
    if Bildirim:
        print 'OrnekFonk1 çalışıyor.'
```

Bu kodun kayıtlı olduğu betik dosyasını açıp F5 ile Python Shell ekranına gelelim:

```
>>> OrnekFonk1()
OrnekFonk1 çalışıyor.
```

Anlaşıldığı gibi, fonksiyonların dışında tanımlanan değişkenler, fonksiyonların içinden görülebilirler. İkinci örneğimizi inceleyelim:

```
BuFonksiyonCagrildi = False
def OrnekFonk2():
    BuFonksiyonCagrildi = True
```

Bu kodun kayıtlı olduğu betik dosyasını açıp F5 ile Python Shell ekranına gelelim:

```
>>> OrnekFonk2()
>>> BuFonksiyonCagrildi
False
```

Burada, fonksiyonun içerisinde "BuFonksiyonCagrildi" isminde yeni bir değişken oluşturulduğu için, aynı isme sahip olan ancak fonksiyonun dışında bulunan "BuFonksiyonCagrildi" değişkeninin değeri olduğu gibi kalmıştır. Yani, "BuFonksiyonCagrildi" isminde 2 değişken vardır ve bunlardan fonksiyonun dışında olanı, fonksiyon hariç her yerde geçerlidir. Fonksiyonun içinde olanı ise sadece fonksiyonun içinde geçerlidir. Haliyle, **True** değerini fonksiyonun içindeki "BuFonksiyonCagrildi" değişkenine atadığımız için ve de Python Shell' de ise komut satırına "BuFonksiyonCagrildi" yazdığımızda Python yorumlayıcısı dışarıda kalan "BuFonksiyonCagrildi" değişkenini anladığı için ekranda **False** değeri görünecektir.

Fonksiyonun içinde, "Hayır, ben dışarıdaki değişkeni kastediyorum, fonksiyonun içinde aynı isimde yeni bir değişken oluşturma." demek için, "**global**" komutu kullanılır. Aşağıdaki betik dosyasını inceleyelim:

```
BuFonksiyonCagrildi = False
def OrnekFonk3():
    global BuFonksiyonCagrildi
    BuFonksiyonCagrildi = True
```

Bu kodun kayıtlı olduğu betik dosyasını açıp F5 ile Python Shell ekranına gelelim:

```
>>> OrnekFonk3()
>>> BuFonksiyonCagrildi
True
```

Tuple Veri Türü

Sıralanmış değerler topluluklarıdır. Gösterimi aşağıdaki gibi olup, tek bir elemana sahip olsa bile bu elemandan sonra virgül konmalıdır:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
('a', 'b', 'c', 'd', 'e')
>>> t1 = ('a',)
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
>>> t2 = ('a')
>>> type(t2)
<type 'str'>
>>> p = tuple('python')
>>> print p
('p', 'y', 't', 'h', 'o', 'n')
>>> print p[1]
y
>>> print p[3 : 5]
('h', 'o')
>>> p[4] = 'j'
***Hata Mesajı (Tuple veri türü, atamayı desteklemiyor.)***
>>> u = ('f',) + p[1:]
>>> print u
('f', 'y', 't', 'h', 'o', 'n')
```

İki değişkenin değerlerini değiş - tokuş yapmak için tuple yapısı büyük kolaylık sağlar:

```
>>> a = 3
>>> b = 7
>>> a, b = b, a
>>> a
7
>>> b
3
```

"**split**" komutu kullanarak örneğin, bir elektronik posta adresini "@" işaretinden öncesi ve sonrası olmak üzere ikiye ayırma işlemini rahatlıkla yapabiliriz:

```
>>> adres = 'python-help@python.org'
>>> KullaniciAdi, AlanAdi = adres.split('@')
>>> print KullaniciAdi
python-help
>>> print AlanAdi
python.org
```

Bazı fonksiyonların geri döndürdüğü değerler 'tuple' türünden olabilir. Örneğin "**divmod**" fonksiyonu bir tamsayıyı diğerine böldüğünde sonuç olarak hem bölümü hem de kalanı döndürür:

```
>>> sonuc = divmod(7, 2)
>>> print sonuc
(3, 1)
>>> bolum, kalan = divmod(7, 2)
>>> print bolum
3
>>> print kalan
1
```

Peki, bu şekilde değer döndüren fonksiyonların tanımlanması nasıl olmalı? Aşağıdaki betik dosyasında yer alan ve verilen bir tamsayının karesini ve karekökünü döndüren fonksiyonu inceleyelim:

```
def KareVeKarekok(tamsayi):
    if ((type (tamsayi) == type (1)) & (tamsayi >= 0)):
        return tamsayi**2, tamsayi**0.5
    else:
        print 'Lütfen pozitif bir tamsayı giriniz.'
```

F5 ile Python Shell ekranına geldiğimizde:

```
>>> KareVeKarekok(16)
(256, 4.0)
>>> kare, karekok = KareVeKarekok(25)
>>> print kare
625
>>> print karekok
5.0
```

Fonksiyonların değişken sayıda argüman almasını sağlamak için "*" işareti kullanılır.

```
def HepsiniYazdir(* argumanlar):
    print argumanlar
```

Bu betik dosyasını çalıştırırsak:

```
>>> HepsiniYazdir(5, 'klm', 7.0, "pqr")
(5, 'klm', 7.0, 'pqr')
```

Yukarıdaki fonksiyona daha fazla sayıda argüman verilse de hepsini 'tuple' şeklinde ekrana yazdıracaktır.

Benzer biçimde, birden fazla sayıda argüman alan bir fonksiyona, içerisinde o fonksiyonun alacağı sayıda değer bulunduran bir 'tuple' ı argüman olarak vermek için gene "*" işareti kullanılır. İki argüman alan **divmod** fonksiyonuna argüman olarak 'tuple' geçirilmesi ile ilgili örneği inceleyelim:

```
>>> t = (7, 3)
>>> divmod(t)
***Hata Mesajı (divmod, tuple'ı tek argüman gibi görüyor)***
>>> divmod(*t)
(2, 1)
```

Tuple' lar, büyüklük-küçüklük, eşitlik yönünden karşılaştırılabilirler. İki 'tuple' karşılaştırılırken ilk önce birinci elemanlarına bakılır. Birinci elemanı diğerinden büyük olan 'tuple', diğerinden daha büyüktür (Geri kalan elemanlara bakılmaz.). Eşitlik halinde ise 2. elemana, 3. elemana... bakılır:

```
>>> (0, 1, 99) < (0, 2, 0)
True
>>> (1, 2, 3) < (1, 1, 77)
False
```

Alıştırmalar

Alıştırma – 1

Görev

Bir karakter dizisi içerisinde her bir karakterden kaç tane bulunduğunun çıkarılması işlemine "histogram çıkarma" denir. Argüman olarak bir karakter dizisi alan ve sonuç olarak ekrana, bu dizi içerisinde hangi karakterden kaçar tane bulunduğunu yazdıran "Histogram" isminde bir fonksiyon hazırlayarak "Histogram.py" isimli betik dosyasına kaydediniz. Programın çalıştırılmasına dair örnek aşağıdadır:

```
>>> Histogram('sekiz yüz seksen sekiz')
  : 3
e : 4
i : 2
k : 3
n : 1
s : 4
y : 1
z : 3
ü : 1
```

NOT: Yukarıdaki kutunun ikinci satırında, boşluk karakterinden 2 tane yer aldığı anlaşılmaktadır.

İpucu

"**for** ***[karakter]*** **in** ***[karakter dizisi]***" kalıbını ve **sözlük** veri türünü bir arada kullanmayı deneyiniz.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 2

Görev

Argüman olarak bir sözlük ve değer alan, sonuç olaraksa o değerin yer aldığı **tüm indeksleri** ekrana yazdıran (eğer o değer hiçbir indekste yer almıyorsa ekrana "İndeks bulunamadı." yazdıran) ve değer döndürmeyen bir fonksiyon hazırlayınız. Fonksiyonunuza "TersArama" adını vererek "TersArama.py" adlı betik dosyasına kaydediniz. Fonksiyonun kullanılmasına dair örnek aşağıdadır:

```
>>> Sehirler = {'Samsun': 'Karad.', 'Antalya': 'Akd.',
                'Tokat': 'Karad.', 'Manisa': 'Ege'}
>>> TersArama(Sehirler, 'Karad.')
Samsun
Tokat
>>> TersArama(Sehirler, 'Akd.')
Antalya
>>> TersArama(Sehirler, 'Ege')
Manisa
>>> TersArama(Sehirler, 'Manisa')
İndeks bulunamadı.
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 3

Görev

"**0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 ...**" şeklinde devam eden ve her elemanın, kendisinden önceki iki elemanın toplamından oluştuğu kurallı diziye "**Fibonacci dizisi**" denir (Bu dizinin sıfırıncı elemanı "0", yedinci elemanını "13", onuncu elemanını "55" olarak düşünelim.). "**Fibonacci**" adında bir fonksiyonumuz olduğunu ve argüman olarak negatif olmayan bir tamsayı alıp, sonuç olarak Fibonacci dizisinin o tamsayıya karşılık gelen indeksindeki elemanı döndürdüğünü düşünürsek, "**Fibonacci(11)**" bize **89** değerini verecektir. **Fibonacci** fonksiyonunun sonucu hesaplamak için yapacağı işlemler, argüman olarak aldığı tamsayı büyüdükçe artacaktır. Bunun için, performans açısından belli bir değere kadarki tamsayıların Fibonacci karşılıklarının program içerisinde tutulmasında yarar vardır (küçük değerlerin daha sık kullanıldığını düşünürsek).

10 ve 10' dan küçük tamsayılar (argüman olarak fonksiyona verilen) için toplama ve benzeri hesaplama işlemi yapmadan direkt sonuç döndüren, 10' dan büyük tamsayılar için hesaplama yaparak sonuç döndüren bir Fibonacci fonksiyonu oluşturunuz. Fonksiyonunuza "HizliFibonacci" ismini vererek "HizliFibonacci.py" isimli betik dosyasına kaydediniz. Fonksiyonun çalıştırılmasına ait örnek aşağıdadır:

```
>>> HizliFibonacci(7)
13
>>> HizliFibonacci(11)
89
>>> HizliFibonacci(19)
4181
>>> HizliFibonacci(0)
0
>>> HizliFibonacci(-7)
Lütfen pozitif bir tamsayı giriniz.
>>> HizliFibonacci(12.7)
Lütfen pozitif bir tamsayı giriniz.
```

İpucu

"Sözlük" başlığını inceleyiniz.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 4

Görev

Sınırsız sayıda sayının tek bir fonksiyon çağrısı ile toplanabilmesi için, sınırsız sayıda argüman kabul ederek aldığı sayıları toplayan ve ekrana toplamı yazdıran bir fonksiyon hazırlayınız. Fonksiyonunuza "SinirsizToplam" ismini vererek "SinirsizToplam.py" isimli betik dosyasına kaydediniz. Fonksiyonun çalıştırılmasına ait örnek aşağıdadır:

NOT: Fonksiyona, sadece tamsayı ve ondalıklı sayı girileceği garanti edilmiştir. Ayrıca hata kontrolü yapmanız gerekmemektedir.

```
>>> SinirsizToplam(3, 5.0, 8, -0.1, 0)
15.9
>>> SinirsizToplam(4, -5, 6)
5.0
```

İpucu

"Tuple Veri Türü" bölümünü inceleyiniz.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

LABORATUVAR ÇALIŞMASI – 7

Bu Çalışmanın Amacı

Bu çalışmadaki amacımız; rasgele (random) sayılar, dosya işlemleri ve sınıflar & nesneler konularında öğrendiklerimizi pekiştirmektir.

Rasgele Sayılar

Python' da rasgele sayılar üretmek için "random" modülünden yararlanılır. 0.0 ile 1.0 arasında (0.0 dâhil, 1.0 hariç) rasgele sayılar üretmek için aşağıdaki örnekten faydalanılabilir:

```
>>> import random
>>> for i in range(5):
        x = random.random()
        print x


0.129072521522
0.555421205583
0.0736199615609
0.281250030657
0.472008699756
```

Belirli bir aralıkta (örneğin 5 ile 10 aralığında, 5 ve 10 dahil olmak üzere) rasgele tamsayı üretmek için "**randint**" komutu kullanılır:

```
>>> import random
>>> random.randint(5, 10)
9
>>> random.randint(5, 10)
5
>>> random.randint(5, 10)
9
>>> random.randint(5, 10)
10
>>> random.randint(5, 10)
7
```

Bir diziden rasgele bir eleman seçmek için:

```
>>> import random
>>> t = [1, 2, 3, 4, 5, 6, 7, 8]
>>> random.choice(t)
4
```

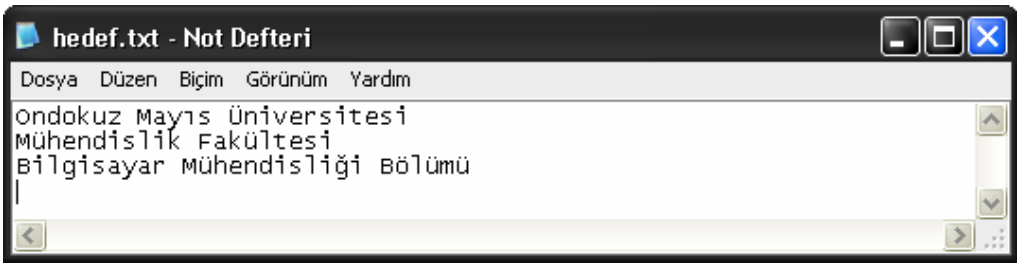
Dosya İşlemleri

Hazırladığımız bir programı çalıştırarak çeşitli veriler elde ettikten sonra bu verileri daha sonra tekrar kullanmak üzere **kalıcı** olarak saklamamız gerekebilir. Bilgisayarı kapatıp açtıktan sonra dahi verilerimizin kaybolmaması için onları **dosyalara** kaydederek sabit diskte saklamamız gerekir.

Python çalışma dizini içerisinde (Python Shell ekranı açıkken Ctrl+O ile görüntülediğimiz dizinde) "hedef.txt" adında bir dosya oluşturarak birinci satırına "Ondokuz Mayıs Üniversitesi", ikinci satırına "Mühendislik Fakültesi", üçüncü satırına ise "Bilgisayar Mühendisliği Bölümü" yazdırarak kaydetmek için şu işlemleri yapmamız gerekir:

```
>>> satir1 = 'Ondokuz Mayıs Üniversitesi\n'
>>> satir2 = 'Mühendislik Fakültesi\n'
>>> satir3 = 'Bilgisayar Mühendisliği Bölümü\n'
>>> fout = open('hedef.txt', 'w')
>>> fout.write(satir1)
>>> fout.write(satir2)
>>> fout.write(satir3)
>>> fout.close()
```

Buradaki "**fout**" dosya nesnesini tutan değişken olup, dosya işlemlerinin yapılması için gereklidir. "fout" yerine istediğimiz herhangi bir ismi de verebiliriz ancak bir kodlama standardı oluşturmak adına "fout" adını kullanmak daha uygun olacaktır. Karakter dizilerinin sonlarındaki "**\n**" karakterleri, metin dosyasında bir alt satıra geçmek içindir. "**open**" fonksiyonu ile dosyayı yazma işlemi için ('w' parametresi yazma işlemini ifade eder.) oluşturduktan (ilk parametrede verilen dosya ismi ile) sonra bu fonksiyonun döndürdüğü nesneyi "**fout**" değişkenine atarız. Bu nesne üzerinde "**write**" fonksiyonunu kullanarak istediğimiz karakter dizilerini dosyaya yazdırırız. Son olarak ise "**close**" fonksiyonu ile dosya nesnesinin görevini sonlandırırız (Bunu yapmazsak veriler dosyaya kaydedilmez [Ancak Python' u kapattığımızda kaydedilir.] ve çeşitli sorunlarla karşılaşabiliriz.). Oluşan dosyayı açtığımızda yazılanları görebiliriz (Windows için görünüm):



Dosyaya yazdığımız veriler **karakter dizisi olmak zorundadır**. Bu nedenle, herhangi bir işlemin sonucunda elde ettiğimiz sayı değerlerini dosyaya yazdırmamız gerektiğinde, öncelikle "**str()**" fonksiyonunu kullanarak bunları karakter dizisine dönüştürmeliyiz:

```
...
>>> x = 0.362
>>> type(x)
<type 'float'>
>>> fout.write(str(x))
...
```

Python' da dosya işlemleri üzerinde çalışırken disk üzerinde çeşitli klasörlere gitmemiz gerekebilir. Bu tür işlemler için "**os**" modülünden faydalanabiliriz (os: operating system [işletim sistemi]). Aşağıdaki örnek kullanımları inceleyelim:

```
>>> import os
>>> CalismaDizini = os.getcwd()
>>> print CalismaDizini
C:\Python25
```

Burada, "**getcwd**" fonksiyonu ile Python' un şu anki çalışma dizini elde edilmiştir. Yani, Python' da bir dosya oluşturarak, adres vermeden (sadece dosya ismi ile) kaydedecek olursak dosya bu dizine kaydedilecektir (Windows işletim sistemi için geçerli).

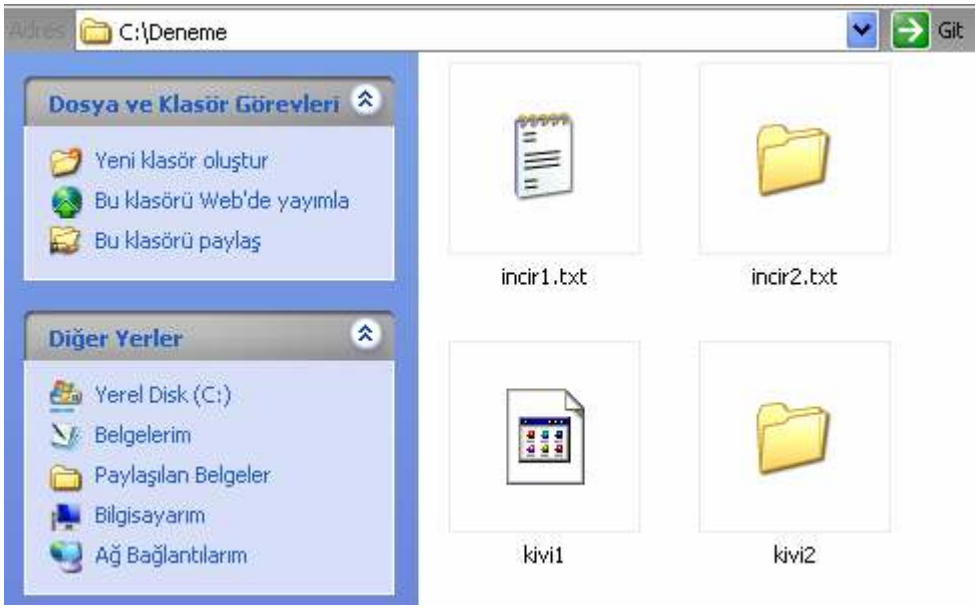
Python' da iki tür dosya yolu vardır. Bunlardan birincisi **tam adres** (**absolute path**) olup, dosyanın adresini kök dizininden itibaren verir (Windows' ta *"C:\Documents and Settings\...\...\hedef.txt"* gibi). Diğeri ise **bağıl adres** (**relative path**) olup, şu anki çalışma dizininden sonrasını verir (Şu anki çalışma dizinimiz *"C:\Python25"* ise ve bir dosyanın bağıl adresi *"Deneme\hedef.txt"* ise bu dosyanın **tam adresi** *"C:\Python25\Deneme\hedef.txt"* olacaktır.).

Çalışma dizinimizde bulunmakta olan "hedef.txt" dosyasının tam adresi şu şekilde elde edilebilir:

```
>>> import os
>>> TamAdres = os.path.abspath('hedef.txt')
>>> print TamAdres
C:\Python25\hedef.txt
```

Bir dosyanın ya da klasörün var olup olmadığı "**exists**" fonksiyonu ile öğrenilebilir (Windows için bir örnek):

```
>>> import os
>>> os.path.exists('C:\Documents and Settings')
True
>>> os.path.exists('C:\Settings and Documents')
False
>>> os.path.exists('C:\hiberfil.sys')
True
>>> os.path.exists('C:\hiberfil.txt')
False
```



Bir adresin, bir dosyaya mı yoksa bir klasöre mi ait olduğunu anlamak gerektiğinde (yandaki örnek için) "**isdir**" fonksiyonunu kullanabiliriz:

```
>>> import os
>>> os.path.isdir('C:\Deneme\incir1.txt')
False
>>> os.path.isdir('C:\Deneme\incir2.txt')
True
>>> os.path.isdir('C:\Deneme\kivi1')
False
>>> os.path.isdir('C:\Deneme\kivi2')
True
```

Herhangi bir dizinin (klasörün) içerisindeki tüm dosya ve klasörleri listelemek için "**listdir**" fonksiyonu kullanılır. Ancak, klasörlerin içerisindeki dosya ve klasörleri listelemez.

```
>>> import os
>>> os.listdir('C:\Python25\libs')
['bz2.lib', 'libpython25.a', 'pyexpat.lib', 'python25.lib',
'select.lib', 'unicodedata.lib', 'winsound.lib',
'_bsddb.lib', '_ctypes.lib', '_ctypes_test.lib',
'_elementtree.lib', '_hashlib.lib', '_msi.lib',
'_socket.lib', '_sqlite3.lib', '_ssl.lib', '_testcapi.lib',
'_tkinter.lib']
```

Hata Yakalama

Dosya işlemlerinde ve pek çok işlemde, yazdığımız program hatalarla karşı karşıya kalabilir. Örneğin dosya işlemlerinde, bir dosyadan okuma yapmak için kullanılan "**open**" fonksiyonuna parametre olarak olmayan bir dosyanın adını verirsek Python yorumlayıcısı hata mesajı verir. Bu tip hatalardan kurtulmak için "exists", "isdir" gibi fonksiyonlarla kontroller koysak bile nerede, ne zaman, hangi hata ile karşılaşacağımızı kestiremeyebiliriz. Bunun için "**try...except...**" bloklarını kullanabiliriz:

```
>>> try:
        fin = open('OlmayanBirDosya.txt')
        for line in fin:
                print line
        fin.close()
except:
     print 'Bir hata oluştu.'


Bir hata oluştu.
```

"try" ile "except" arasında ne tür bir hata meydana gelirse gelsin program, ekrana "Bir hata oluştu" yazdırmaktan başka bir şey yapmayacaktır. Böylece programın geri kalanı çalışmaya devam edecektir.

Veritabanı Kullanımı

Python' da veritabanı kullanımı, sözlük kullanımına benzemekte olup, "**anydbm**" modülülnden yararlanılmaktadır. Örnek kullanım aşağıdadır:

```
>>> import anydbm
>>> VeriTabani = anydbm.open('ornekVT.db', 'c')
>>> VeriTabani['universite'] = 'Dokuz Eylül Üniversitesi'
>>> print VeriTabani['universite']
Dokuz Eylül Üniversitesi
>>> VeriTabani['universite'] = 'Ondokuz Mayıs Üniversitesi'
>>> print VeriTabani['universite']
Ondokuz Mayıs Üniversitesi
>>> VeriTabani['kent'] = 'Samsun'
>>> for Anahtar in VeriTabani:
        print Anahtar


universite
kent
>>> VeriTabani.close()
```

İkinci satırda gördüğümüz "**'c'**" komutu, yeni bir veritabanı oluşturulması içindir. Veritabanını kullandıktan sonra, son satırda olduğu gibi kapatmalıyız.

Modül Oluşturma

İçerisinde Python kodu bulunduran herhangi bir dosya, **modül** olarak çağrılabilir. Çağırma işlemi için "**import**" komutu kullanılır. Modül çağrıldıktan sonra, içerisinde yer alan tüm fonksiyonlar tanımlanmış, tüm komutlar ise çalıştırılmış olacaktır. Aşağıdaki kodu, çalışma dizini (Python25 klasörü) içerisinde "SatirSay.py" isimli betik dosyasına kaydedelim:

```
def SatirSay(DosyaAdi):
    sayac = 0
    for Satir in open(DosyaAdi):
        sayac += 1
    return sayac

print SatirSay('sozcuk.txt')
```

PythonShell ekranına gelerek şunları yazalım:

```
>>> import SatirSay
8
```

Bu işlemle hem fonksiyon tanımlanmış, hem de modülün son satırında yer alan komut ile tanımlanan fonksiyon çalıştırılarak sonuç elde edilmiştir.

Sınıflar ve Nesneler

Python içerisinde yer alan veri türlerinin (karakter dizisi, tamsayı, 'boolean' gibi) dışında, kendi belirlediğimiz ve adlandırdığımız yeni veri türleri tanımlamamız da mümkündür. Bunun için önce bir **sınıf** tanımlayıp, daha sonra bu **sınıfın bir nesnesini** oluşturmamız gerekir. İşte bahsi geçen bu nesnenin **türü**, tanımladığımız **sınıf** olacaktır.

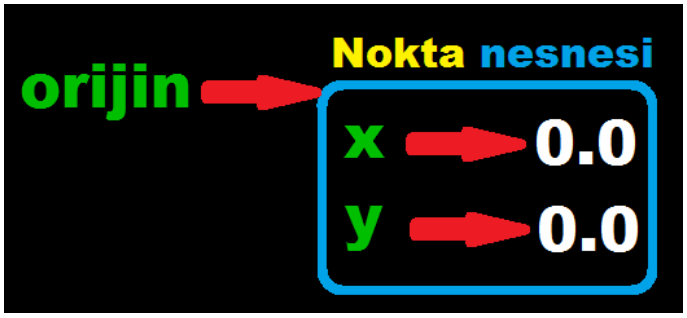
Geometride noktaların x ve y koordinatları ile tutulduklarını göz önünde bulundurarak oluşturulmuş "**Nokta**" sınıfını ve bu sınıfın bir nesnesi olan, başka bir deyişle **Nokta** türünden olan nesnelerin ele alındığı aşağıdaki örneği inceleyelim (Python Shell üzerinde):

```
>>> class Nokta(object):
        """Düzlemdeki bir noktayı x ve y koordinatları ile
                                                ifade eder."""
```

Yukarıda "Nokta" adında bir **sınıf** tanımlanmış olup, tırnaklar arasındaki ifade sınıf hakkında açıklama niteliğindedir. Tanımlamış olduğumuz bu sınıfın **tam ismi** "__main__.Nokta" dır. Şimdi bu sınıfın bir nesnesini oluşturalım:

```
>>> orijin = Nokta()
>>> orijin.x = 0.0
>>> orijin.y = 0.0
>>> type(orijin)
<class '__main__.Nokta'>
```

Yukarıdaki kutuda **Nokta** sınıfının bir nesnesi oluşturulmuş olup, bu nesnenin ismi "orijin" dir. "orijin" nesnesine ise "**x**" ve "**y**" isimlerinde alt alanlar (**attribute**) eklenmiş olup, bu alt alanlarda ondalıklı sayı türünden koordinat değerleri tutulmaktadır. Elde edilen yapıyı ifade eden görsel bilgi aşağıdadır:



“orijin” nesnesi üzerinden, bu nesnenin alt alanları olan “x” ve “y” ye ulaşmak mümkündür:

```
>>> orijin.x
0.0
>>> print orijin.y
0.0
>>> type(orijin.x)
<type 'float'>
```

Şimdi ise, argüman olarak **Nokta** nesnesi alan ve ekrana, bu nesne içerisinde koordinatları tutulan noktayı “(a, b)” formatında yazan bir fonksiyon tanımlayalım (Python Shell ekranında):

```
>>> def  NoktayiYazdir(nkt):
    print '(', nkt.x, ',', nkt.y, ')'


>>> NoktayiYazdir(orijin)
( 0.0 , 0.0 )
```

Fonksiyonlar, değer olarak **nesne** de döndürebilirler. Aşağıdaki betik dosyası örneğinde, kullanıcıdan iki farklı noktanın koordinatını alarak bu noktaları birleştiren doğru parçasının orta noktasını veren bir fonksiyon ele alınmıştır:

```
class Nokta(object):
    """Düzlemdeki bir noktayı x ve y koordinatları ile ifade
                                                    eder."""

def OrtaNokta():
    nokta1_x = int(raw_input("Birinci noktanin x
                                koordinatini giriniz : "))
    nokta1_y = int(raw_input("Birinci noktanin y
                                koordinatini giriniz : "))
    nokta2_x = int(raw_input("Ikinci noktanin x koordinatini
                                        giriniz  : "))
    nokta2_y = int(raw_input("Ikinci noktanin y koordinatini
                                        giriniz  : "))

    merkez = Nokta()
    merkez.x = (nokta1_x + nokta2_x) / 2.0
    merkez.y = (nokta1_y + nokta2_y) / 2.0

    return merkez
```

F5 ile Python Shell ekranına geçtiğimizde:

```
>>> N = OrtaNokta()
Birinci noktanin x koordinatini giriniz : 2
Birinci noktanin y koordinatini giriniz : 4
Ikinci noktanin x koordinatini giriniz  : 5
Ikinci noktanin y koordinatini giriniz  : 8
>>> N
<__main__.Nokta object at 0x00C953D0>
>>> N.x
3.5
>>> N.y
6
```

Görüldüğü gibi "OrtaNokta" fonksiyonu, değer olarak **Nokta** türünden bir nesne döndürmektedir.

Print Komutunda Parametre Kullanımı

Print komutu ile değişkenlerin değerlerini yazdırmak istediğimizde bunu tırnakları defalarca açıp - kapatmaya gerek duymadan yapmamız mümkündür. Tırnak içerisindeki karakter dizisinde; tamsayı değerine sahip bir değişkenin değerini yazmak için "%d", ondalıklı sayı değerine sahip bir değişkenin değerini yazmak için "%g", karakter dizisi değerine sahip bir değişkenin değerini yazmak içinse "%s" ifadelerini koymalı, yazacaklarımızı bitirdikren ve tırnağı kapattıktan sonra da "%" işareti koyarak parantez açmalı ve ilgili değişkenleri sırasıyla, aralarına virgül koyarak yazmalıyız:

```
>>> ad = 'Damla'
>>> yas = 25
>>> boy = 1.65
>>> print '%s, %d yaşında ve %g boyundadır.' % (ad, yas, boy)
Damla, 25 yaşında ve 1.65 boyundadır.
>>> print '%s, %d yaşında ve %g boyundadır.' % (boy, yas, ad)
***Hata Mesajı***
```

Eğer sıralamaya dikkat etmezsek hata mesajlarıyla karşılaşabileceğimiz gibi, hata mesajı vermeyen ancak yanlış çıktı veren bir programla da karşılaşabiliriz.

Alıştırmalar

Alıştırma – 1

Görev

0.0 ile 1.0 arasında (0.0 dâhil, 1.0 hariç) rasgele bir sayı seçildiğinde, teorik olarak 1' e daha yakın olma ihtimali ile 0' a daha yakın olma ihtimali aynıdır. Bu bilgiden hareketle, 0.0 ile 1.0 arasında **n** tane rasgele sayı üretip, bu sayıları toplayıp **n**' ye bölerek ortalamasını alınız. Bu işlemi n = 10, n = 25, n = 50, n = 100, n = 250, n = 500, n = 1000 ve n = 5000 için yaparak çıkan sonuçları (elde edilen ortalamaları) gözleyiniz. n sayısı büyüdükçe ortalama belli bir değere yaklaşıyor mu yoksa ortalama da tamamen 0 ile 1 arasında rasgele değerler mi alıyor? Size göre bunun açıklaması nedir? Yorumlayınız.
NOT: Bu problemin çözümünde bulunması gereken tek bir cevap yoktur.

İpucu

"Rasgele Sayılar" bölümünü inceleyiniz.

Sonuç

Gerçekleştiriminizi, yorumlarınızı ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 2

Görev

Kendi içerisinde yazan her şeyi, kendisi ile aynı dizinde bulunan "kopya.txt" isimli dosya içerisine yazdıran bir betik dosyası hazırlamayı deneyiniz (Betik dosyanıza "KaynakKod.py" ismini veriniz.). Yani, betik dosyanızı düzenleyip F5 tuşuna bastığınızda, betik dosyanızla aynı dizinde "kopya.txt" isminde bir dosya oluşmalı, bu dosyanın içini açıp baktığınızda da birebir betik dosyanızda yer alan kodlarla karşılaşmalısınız. Başarabildiniz mi? Sonuçları yorumlayınız.

NOT: Dosya kopyalama fonksiyonu kullanmamanız gerekmektedir.

İpucu

"Dosya İşlemleri" ve "Laboratuar Çalışması 5 - Dosyadan Veri Okuma" bölümlerini inceleyiniz. Bu problem, 6 satırlık Python kodu ile çözülebilmektedir.

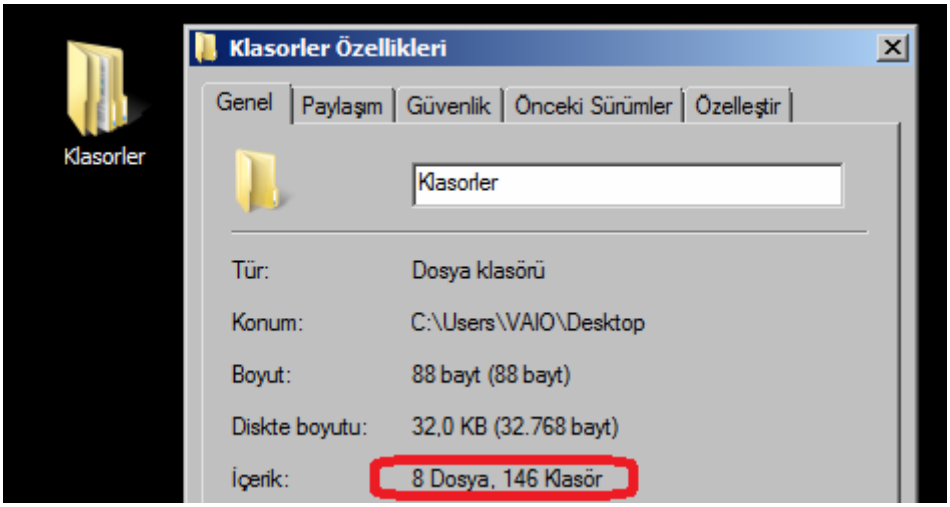
Sonuç

Gerçekleştiriminizi, yorumlarınızı ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 3 (ÖDÜLLÜ SORU)

Görev

Size, "Klasorler" isminde, önceden hazırlanmış bir klasör verilecektir. Bu klasörün içinde de pek çok sayıda iç içe klasör bulunmaktadır. Bu iç içe klasörlerin bazılarının içerisinde de dosyalar bulunmaktadır. Sizden, "Klasorler" klasörünün içerisinde toplam kaç klasör ve kaç dosya bulunduğunu hesaplamanız istenmektedir."Klasorler" klasörü, içerisinde 146 klasör ve 8 dosya bulundurmaktadır:



Hazırlayacağınız program, "Klasorler" isimli klasör de dahil olmak üzere toplamda **147 klasör** ve **8 dosya** olduğu sonucunu vermelidir. "DosyaKlasorSay.py" isminde bir betik dosyasına kaydetmeniz gereken kodlarınız, betik dosyası açıkken F5 tuşuna bastığınızda şu çıktıyı vermelidir:

```
Klasör sayısı : 147
Dosya sayısı  : 8
```

Betik dosyanızda fonksiyonlar, fonksiyon çağrıları, fonksiyonların dışarısında yapılan (dosyadan veri okuma vs.) işlemler bulunabilir. Bu problemin çözümü için, bu kaynağın dışına çıkarak, internet üzerinde ve diğer kaynaklarda araştırmalar yapmanız da gerekmektedir. Problem üzerinde dikkatlice düşünmeniz, karşılaşabileceğiniz olumsuz durumlar karşısında ise çok daha değişik taktiklere başvurmanız beklenmektedir. Python programlama dili kullanmak şartı ile istediğiniz yönteme başvurabilirsiniz. Bu alıştırmada, araştırma yeteneğinizin de ölçüldüğünü göz önünde bulundurunuz.

İpucu

“os.path.walk()” fonksiyonunu, işleyiş biçimi ve aldığı parametreler yönünden araştırınız.

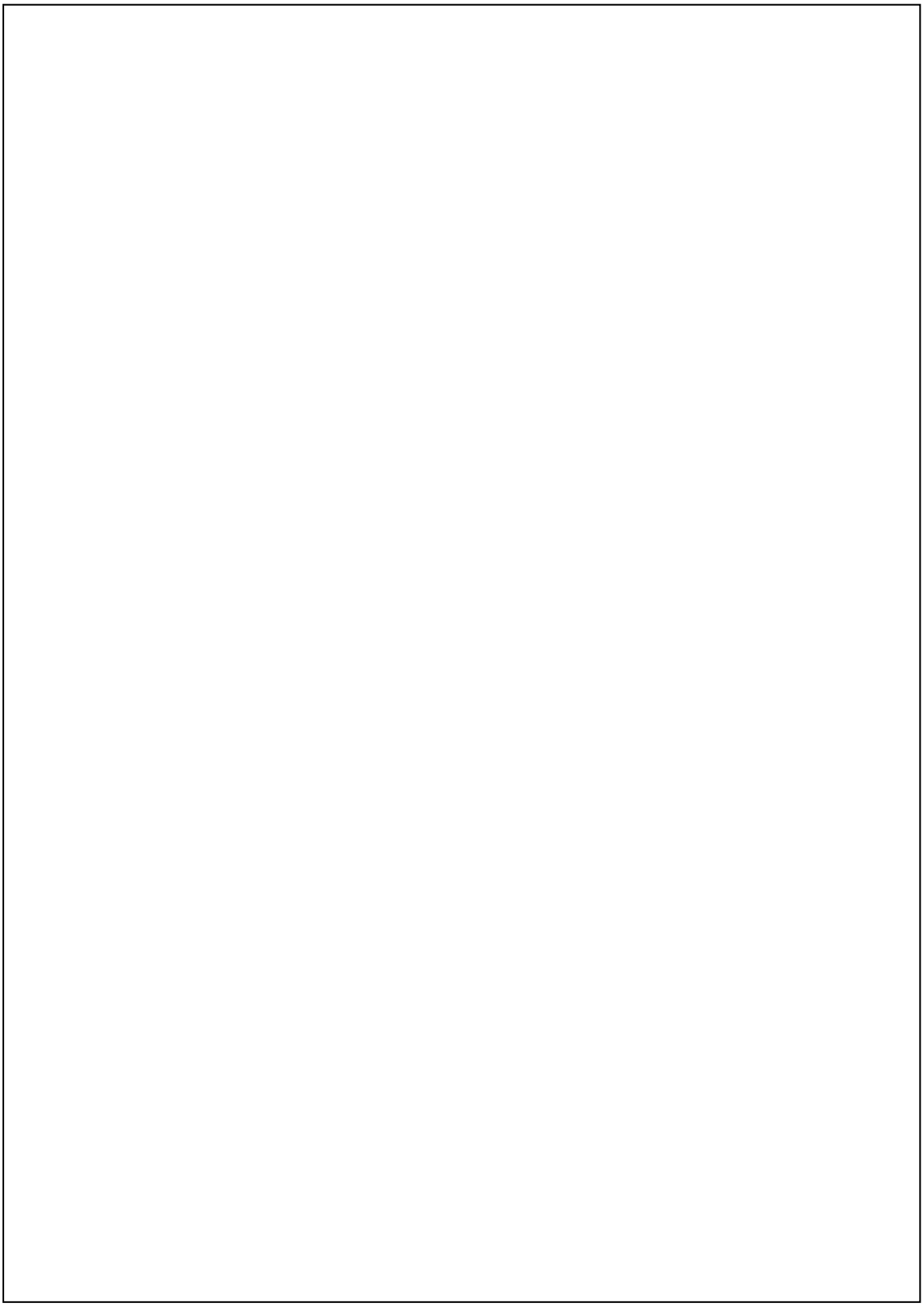
Eğer kullanmanız gereken bazı fonksiyonları her çağırışınızda tutmakta olduğunuz verilerin (dosya ve klasör sayısı gibi) sıfırlanması ya da silinmesi problemi ile karşılaşıyorsanız, her işleminizin sonucunda metin dosyasına gerekli verileri kaydetmek iyi bir fikir olabilir. Dosyaya yazma ile ilgili örneğimizde, “open” komutunun, ikinci parametre değeri “w” idi; “w” haricinde ne tür değerler alabileceğini de araştırmanız yararınıza olabilir (Eğer problemi çözmek için dosyaya yazma ve dosyadan okumaya ihtiyaç duyarsanız.).

Bir adresin sonuna (“C:\Python25” gibi) bir dosya adı (“metin.txt”) ekleme işlemi için yardımcı olabilecek olan aşağıdaki örneği inceleyiniz:

```
>>> adres = 'C:\Python25'
>>> dosya = 'metin.txt'
>>> TamYol = adres + '\\' + dosya
>>> TamYol
'C:\\Python25\\metin.txt'
```

Sonuç

Gerçekleştiriminizi, kullandığınız yöntemin mantığını, yorumlarınızı ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız. Programlama ile çözüm üretememiş olsanız bile çözmek için hangi yöntemi kullanmak istediğinizi yazınız. Yararlandığınız kaynakları da belirtmeyi unutmayınız.



LABORATUVAR ÇALIŞMASI – 8

Bu Çalışmanın Amacı

Bu çalışmadaki amacımız, Python' daki sınıflar ve fonksiyonlar hakkında öğrendiklerimizi pekiştirmektir.

Sınıflar ve Fonksiyonlar

Python' da kullanıcılar sınıf ve fonksiyon tanımlaması yapabilirler. Bunu; zamanı saat, dakika ve saniye cinsinden tutmak ve tutulan bu değerler üzerinde çeşitli işlemler yapmak için hazırlanmış olan bir örnek yardımıyla ele alacağız:

```
>>> class ZamanSinifi(object):
        """Gün içerisinde zamanı temsil eder.
        Alt alanlar: saat, dakika, saniye"""

>>> zaman = ZamanSinifi()
>>> zaman.saat = 10
>>> zaman.dakika = 44
>>> zaman.saniye = 31
                                                    >>devam>>
```

Yukarıdaki kutuda yapılan işlemler sonucunda elde edilen tutacak, nesne ve alt alanların temsil edildiği şekil aşağıdadır:

Saat, dakika ve saniye türünden iki farklı zaman değerini birbiri ile toplayan "ZamanTopla" isimli bir fonksiyon tanımlayarak devam edelim:

```
>>devam>>
>>> def ZamanTopla(z1, z2):
      toplam = ZamanSinifi()
      toplam.saat = z1.saat + z2.saat
      toplam.dakika = z1.dakika + z2.dakika
      toplam.saniye = z1.saniye + z2.saniye
      if toplam.saniye >= 60:
            toplam.saniye -= 60
            toplam.dakika += 1
      if toplam.dakika >= 60:
            toplam.dakika -= 60
            toplam.saat += 1
      return toplam


>>> baslangic = ZamanSinifi()
>>> baslangic.saat = 7
>>> baslangic.dakika = 13
>>> baslangic.saniye = 51
>>> sure = ZamanSinifi()
>>> sure.saat = 2
>>> sure.dakika = 49
>>> sure.saniye = 56
>>> bitis = ZamanTopla(baslangic, sure)
>>> bitis.saat
10
>>> bitis.dakika
3
>>> bitis.saniye
47
```

Nesnelerin Yazdırılması

"ZamanSinifi" isminde bir sınıf tanımlamıştık ve bu sınıfa ait nesneler oluşturmuştuk. Bu nesnelerin alt alanlarında ise saat, dakika ve saniye bilgilerini tutmaktaydık. Şimdi ise, "ZamanSinifi" sınıfının dışarısında, bu sınıfa ait bir nesnenin ekrana "saat:dakika:saniye" formatında yazdırılmasını sağlayan "ZamaniYazdir" isimli bir fonksiyon oluşturalım:

```
class ZamanSinifi(object):
    """Gün içerisinde zamanı temsil eder. Alt alanlar: saat,
                                          dakika, saniye"""
def ZamaniYazdir(zaman):
    print '%.2d:%.2d:%.2d' % (zaman.saat, zaman.dakika,
                                          zaman.saniye)
```

NOT: "%.2d" kullandığımız zaman, yazılacak olan tamsayının son iki basamağı alınır. Sayı tek basamaklı ise, onlar basamağına "0" konur.

Yukarıdaki kutuda yer alan kodu bir betik dosyasına kaydedip F5' e bastıktan sonra Python Shell ekranında şu işlemleri yapalım:

```
>>> baslangic = ZamanSinifi()
>>> baslangic.saat = 7
>>> baslangic.dakika = 13
>>> baslangic.saniye = 51
>>> ZamaniYazdir(baslangic)
07:13:51
```

Yukarıdaki örnekte "ZamaniYazdir" fonksiyonu, "ZamanSinifi" sınıfının dışında tanımlandığı için bu fonksiyonu, bir üstteki kutuda olduğu gibi **argüman geçirerek** çağırabiliriz. Eğer bu fonksiyonu, aşağıdaki gibi "ZamanSinifi" sınıfının içinde tanımlarsak,

```
class ZamanSinifi(object):
    """Gün içerisinde zamanı temsil eder. Alt alanlar: saat,
                                            dakika, saniye"""
    def ZamaniYazdir(zaman):
        print '%.2d:%.2d:%.2d' % (zaman.saat, zaman.dakika,
                                              zaman.saniye)
```

bu sınıfın bir metodu haline gelecektir ve gerek fonksiyon çağırır gibi,

```
...
>>> ZamanSinifi.ZamaniYazdir(baslangic)
07:13:51
```

gerekse klasik metod çağrısı şeklinde

```
...
>>> baslangic.ZamaniYazdir()
07:13:51
```

kullanmamız mümkün olacaktır. Daha yaygın bir gösterim biçimi olarak, sınıf içerisinde metod tanımlaması için "**self**" ifadesini kullanılır:

```
class ZamanSinifi(object):
    """Gün içerisinde zamanı temsil eder. Alt alanlar: saat,
                                            dakika, saniye"""
    def ZamaniYazdir(self):
        print '%.2d:%.2d:%.2d' % (self.saat, self.dakika,
                                              self.saniye)
```

“init” Metodu

Sınıfların nesneleri oluşturulduğunda, bu nesnelere ilk değerleri (varsayılan değerleri) atamak için kullanılan (Python tarafından **otomatik olarak** kullanılır.) ve “**__init__**” biçiminde yazılan bu metodun kullanımına dair örnek aşağıda verilmiştir:

```
class ZamanSinifi(object):
    """Gün içerisinde zamanı temsil eder. Alt alanlar: saat,
                                                dakika, saniye"""
    def __init__(self, saat = 0, dakika = 0, saniye = 0):
        self.saat = saat
        self.dakika = dakika
        self.saniye = saniye

    def ZamaniYazdir(self):
        print '%.2d:%.2d:%.2d' % (self.saat, self.dakika,
                                                  self.saniye)
```

Argüman isimlerinin alt alan isimleri ile aynı olması, genel olarak tercih edilen bir kullanımdır. F5 ile Python Shell ekranına geçecek olursak:

```
>>> zmn = ZamanSinifi()
>>> zmn.ZamaniYazdir()
00:00:00
                                                    >>devam>>
```

Eğer bir tane argüman girersek bu, ilk argüman olan “**saat**” in yerine geçer:

```
>>devam>>
>>> zmn2 = ZamanSinifi(6)
>>> zmn2.ZamaniYazdir()
06:00:00
                                                    >>devam>>
```

Benzer biçimde, iki argüman girersek de ilk argüman “**saat**” in yerine, ikincisi ise “**dakika**” nın yerine geçecektir:

```
>>devam>>
>>> zmn3 = ZamanSinifi(6, 8)
>>> zmn3.ZamaniYazdir()
06:08:00
```

"str" Metodu

Herhangi bir sınıfın içerisinde yer alan ve o sınıfın nesnelerinin karakter dizisi şeklindeki gösterimlerini döndüren ve "**__str__**" biçiminde yazılan bu metodun kullanımına dair örnek aşağıda verilmiştir:

```
class ZamanSinifi(object):
    """Gün içerisinde zamanı temsil eder. Alt alanlar: saat,
                                                dakika, saniye"""
    def __init__(self, saat = 0, dakika = 0, saniye = 0):
        self.saat = saat
        self.dakika = dakika
        self.saniye = saniye

    def __str__(self):
        return '%.2d:%.2d:%.2d' % (self.saat, self.dakika,
                                                self.saniye)

    def ZamaniYazdir(self):
        print '%.2d:%.2d:%.2d' % (self.saat, self.dakika,
                                                self.saniye)
```

Python' da herhangi bir nesneyi ekrana yazdırmak için **print** komutunu kullandığımızda **otomatik olarak** o sınıfta yer alan "**__str__**" metodu kullanılır. F5 ile Python Shell ekranına geçecek olursak:

```
>>> zmn4 = ZamanSinifi(5, 12, 13)
>>> print zmn4
05:12:13
```

Operatör Gölgeleme

Python' da "+", "/" gibi operatörlerin kendilerine has kullanım biçimleri ve görevleri vardır. Ancak biz, bu operatörlerin, oluşturduğumuz sınıfın nesneleri üzerinde hanki işlemleri yapacaklarını **yeniden tanımlayarak**, bu operatörlerin varsayılan görevlerini (bahsedilen tanımlama işlemini yapmadan önceki görevlerini) **gölgelemiş** oluruz; yani artık bu operatörler, bizim tanımlamış olduğumuz işlemleri yerine getirirler. "+" operatörünün gölgelenmesine dair örnek bir sınıf gerçekleştirimi, aşağıda verilmiştir:

```
class TamsayiOrnek(object):
    """Ornek gosterim icin olusturulmus basit bir tamsayi
                                                    sinifi"""
    def __init__(self, sayi = 0):
        self.sayi = sayi
    def __add__(self, deger):
        if isinstance(deger, TamsayiOrnek):
            return (self.sayi + deger.sayi)
        else:
            return (str(self.sayi) + str(deger))
```

Yukarıdaki kutuda yer alan sınıf tanımını incelediğimizde, bu sınıfa ait nesnelerin "**sayi**" isminde bir alt alana sahip olduklarını ve bu sınıfa ait nesneler oluşturulduğunda, bu nesnelerin alt alanlarının varsayılan değerlerinin **0**' a eşit olduğunu görürüz. "+" operatörüne yeni bir anlam yüklemek için, bu operatör kullanıldığında ne olacağını, "**def __add__(...)**" şeklinde tanımladığımız fonksiyonun içerisine yazmamız gerekmektedir. Bir nesnenin, bir sınıfa ait olup - olmadığını **True** ya da **False** değerleri döndürerek bildiren "**isinstance**" fonksiyonundan da yararlanıldığı bu gerçekleştirimde, eğer "+" operatörünün sağındaki değer "TamsayiOrnek" sınıfının bir nesnesi ise her iki değerin de "**sayi**" alt alanı toplanarak sonuç döndürülmektedir. Eğer "+" operatörünün sağındaki değer, "TamsayiOrnek" sınıfına ait bir nesne değilse soldaki değer ile sağdaki değer **karakter dizine çevrilerek arka arkaya eklenmekte** ve sonuç döndürülmektedir.

NOT: Python, "+" operatörü için hangi fonksiyonu kullanacağına karar verirken **ilk önce, "+" operatörünün sol tarafındaki değişkenin ait olduğu sınıfın "__add__" isimli bir fonksiyonu olup-olmadığına** bakar. "+" operatörünün sağ tarafındaki değişkenin, fonksiyon seçiminde bir etkisi yoktur, sadece fonksiyona ikinci argüman olarak geçecektir.

Diğer operatörlerin görevlerini yeniden tanımlamakta kullanılması gereken fonksiyonlar için http://docs.python.org/reference/datamodel.html adresinde **3.4.7** numaralı başlık incelenebilir.

Yukarıdaki kutuda yer alan kodu betik dosyasına kaydedip F5' e basarak Python Shell ekranına geçelim:

```
>>> x = TamsayiOrnek(5)
>>> y = TamsayiOrnek(7)
>>> z = x + y
>>> print z
12
>>> a = TamsayiOrnek(3)
>>> b = 9
>>> c = a + b
>>> print c
39
>>> p = 'python'
>>> r = TamsayiOrnek(6)
>>> s = p + r
***Hata Mesajı***
>>> t = r + p
>>> print t
6python
```

Yukarıdaki kutuda hata mesajı ile karşılaşılmıştır. "**p + r**" ifadesinde "+" operatörünün sol tarafında yer alan "p" değişkeninin yer aldığı "**str**" sınıfında"+" operatörü, karakter dizilerini birleştirmekte kullanılmakta olup, bir "str" nesnesi ile bir "TamsayiOrnek" nesnesini birleştirememektedir. Oysa "r + p" ifadesinde "+" operatörünün solunda, "TamsayiOrnek" sınıfına ait bir nesne yer aldığı için, "TamsayiOrnek" sınıfı içerisinde tanımlamış olduğumuz "__add__" fonksiyonundan yararlanılacak ve karakter dizisi türünden olan "6python" değeri elde edilecektir.

Çok Biçimlilik

Bir fonksiyonun farklı farklı sınıflara ait nesnelerle çalışabilirliğine "**çok biçimlilik (polimorfizm)**" denir. Ayrı ayrı fonksiyonlar tanımlamak yerine birden fazla sayıda farklı sınıfa ait nesneleri kabul edecek tek bir fonksiyon hazırlamak, kod tekrarı yapılmaması ve bu fonksiyonun yeniden kullanımı açısından önemlidir. Aşağıdaki örneği inceleyelim:

```
def IlkElemaniDondur(deger):
    if ((type(deger) == type([1, 2])) | (type(deger) ==
                                          type('abc'))):
        return deger[0]
    else:
        print "Bu değer, bu fonksiyonun biçimine uygun
                                                 değildir."
```

F5 ile Python Shell ekranına geçelim:

```
>>> liste = [6, 7, 8, 9]
>>> type(liste)
<type 'list'>
>>> karakter = 'efgh'
>>> type(karakter)
<type 'str'>
>>> tuple1 = ('m', 7, 'n', 8.76)
>>> type(tuple1)
<type 'tuple'>
>>> IlkElemaniDondur(liste)
6
>>> IlkElemaniDondur(karakter)
'e'
>>> IlkElemaniDondur(tuple1)
Bu değer, bu fonksiyonun biçimine uygun değildir.
```

Görüldüğü gibi, "IlkElemaniDondur" fonksiyonu hem listeler ile, hem de karakter dizileri ile çalışabilmektedir (her ne kadar diğer sınıflara ait veri türlerini destekliyor olmasa da). "IlkElemaniDondur" fonksiyonunun, listeler için ve karakter dizileri için ayrı ayrı iki fonksiyon tanımlaması yapılmasına gerek kalmadan her iki farklı veri türünü de desteklemesi, çok biçimliliğe bir örnektir.

Alıştırmalar

Alıştırma – 1

Görev

Aşağıda, "ZamanSinifi" isimli sınıf verilmiştir:

```
class ZamanSinifi(object):
    """Gün içerisinde zamanı temsil eder. Alt alanlar: saat,
                                                dakika, saniye"""
    def __init__(self, saat = 0, dakika = 0, saniye = 0):
        self.saat = saat
        self.dakika = dakika
        self.saniye = saniye

    def __str__(self):
        return '%.2d:%.2d:%.2d' % (self.saat, self.dakika,
                                                self.saniye)

    def ZamaniYazdir(self):
        print '%.2d:%.2d:%.2d' % (self.saat, self.dakika,
                                                self.saniye)
```

Bu sınıfın yer aldığı betik dosyasının içerisine; **(I)** bir "ZamanSinifi" nesnesi içerisinde tutulan zamanı saniyeye çevirip tamsayı olarak döndüren ("ZamanSinifi" nesnesinden saat-dakika-saniye bilgilerini alarak bunları saniyeye dönüştüren ve sonucu döndüren), **(II)** tamsayı olarak verilen saniyeyi **saat-dakika-saniye** cinsinden hesaplayıp bir "ZamanSinifi" nesnesinin içinde döndüren ve **(III)** verilen iki "ZamanSinifi" nesnesinin zaman bilgilerini toplayarak sonucu **saat-dakika-saniye** cinsinden hesaplayarak bir "ZamanSinifi" nesnesi içerisinde döndüren fonksiyonları yazınız. Fonksiyonlara sırası ile "ZamaniSaniyeye", "SaniyeyiZamana" ve "ZamanlariTopla" isimlerini veriniz. Yukarıda verilen sınıfı, yanına eklenmiş olan fonksiyonlarınızla birlikte "ZamanSinifi.py" isimli bir betik dosyasına kaydediniz.

NOT: "ZamanlariTopla" fonksiyonunun bulacağı saat değeri, 24' ten fazla olabilir. Bu durumda gün hesabı yapmanıza gerek yoktur. "**25:51:35**" gibi bir değer bulmanız daha doğru olacaktır. Ayrıca, yukarıdaki kutuda verilen kod normalde 10 satırdan oluşmaktadır (boş satırlar hariç). Kutuya sığmayan satırlar ikiye bölünmüş; ikinci parçaları, birinci parçalarının yer aldığı satırın altındaki satıra sağa dayalı olarak yazılmıştır. "ZamanSinifi" sınıfını oluştururken bunlara dikkat ediniz. Sınıfın tamamını kendinizin kopyalama yapmadan yazması tavsiye edilir.

Gerçekleştirimin çalıştırılmasına dair örnek, aşağıdadır:

```
>>> zmn1 = ZamanSinifi(4, 10, 25)
>>> zmn2 = ZamanSinifi(23, 52, 46)
>>> print zmn1
04:10:25
>>> print zmn2
23:52:46
>>> zmn1_sn = ZamaniSaniyeye(zmn1)
>>> zmn2_sn = ZamaniSaniyeye(zmn2)
>>> print zmn1_sn
15025
>>> print zmn2_sn
85966
>>> zmn3 = SaniyeyiZamana(60300)
>>> print zmn3
16:45:00
>>> zmn3.dakika
45
>>> zmntop = ZamanlariTopla(zmn1, zmn2)
>>> print zmntop
28:03:11
```

İpucu

"LABORATUVAR ÇALIŞMASI - 6 / Tuple Veri Türü" başlığı altında yer alan "divmod" fonksiyonunun kullanımını inceleyiniz.

"ZamanlariTopla" fonksiyonun gerçekleştiriminde "ZamaniSaniyeye" ve "SaniyeyiZamana" fonksiyonlarından yararlanmanız size kolaylık sağlayacaktır.

Ekleyeceğiniz fonksiyonları "ZamaniSaniyeye" sınıfının içerisine yazarsanız bu fonksiyonlar, argüman olarak "ZamaniSaniyeye" nesneleri isteyecek, dolayısı ile de tamsayı kabul etmeyeceklerdir. "SaniyeyiZamana" fonksiyonunun argüman olarak tamsayı alacağını da düşünerek fonksiyon tanımlamalarınızı sınıfın içerisine değil, sınıfın altına ve sınıf ile aynı hizada yapınız.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 2

Görev

Otomobillere ait marka, model, üretim yılı, renk ve fiyat bilgilerini tutmak için kullanılabilecek "OtoBilgi" isimli bir sınıf hazırlayınız. Sınıfınızda, "__init__" ve "__str__" metodlarıyer almalıdır. "__init__" metodu, bu sınıfın yeni bir nesnesi oluşturulduğunda varsayılan alt alan değerlerini şu şekilde atamalıdır:

marka → **'bilinmiyor'**
model → **'bilinmiyor'**
yil → **0**
renk → **'bilinmiyor'**
fiyat → **'bilinmiyor'**

"__str__" metodunun nasıl bir değer döndüreceğini ise aşağıdaki kullanım örneğini inceleyerek bulunuz. Hazırladığınız sınıfı "OtoBilgi.py" isimli betik dosyasına kaydediniz. Sınıfın kullanımına ait örnek aşağıdadır:

```
>>> araba1 = OtoBilgi()
>>> print araba1
Marka : bilinmiyor
Model : bilinmiyor
Yıl   : 0
Renk  : bilinmiyor
Fiyat : 0

>>> araba2 = OtoBilgi('Volvo', 's80', 2003, 'beyaz', 25000)
>>> print araba2
Marka : Volvo
Model : s80
Yıl   : 2003
Renk  : beyaz
Fiyat : 25000
```

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 3

Görev

"Tamsayi4islemYeni" isminde, "**sayi**" isminde bir alt alana sahip olan bir sınıf tanımlayınız. Sınıfınızda, "+", "-", "*" ve "/" operatörlerinin görevlerini, aşağıda verilen işlemleri gerçekleştirecekleri biçimde değiştiriniz:

a + b → **ab**

a - b → **ba**

a * b → a^b

a / b → $\sqrt[b]{a}$

Sınıfınızı ve operatörler için yeniden tanımladığınız fonksiyonlarınızı "Tamsayi4islemYeni.py" isimli betik dosyasına kaydediniz. Sınıfın kullanımına dair örnek gösterim aşağıdadır:

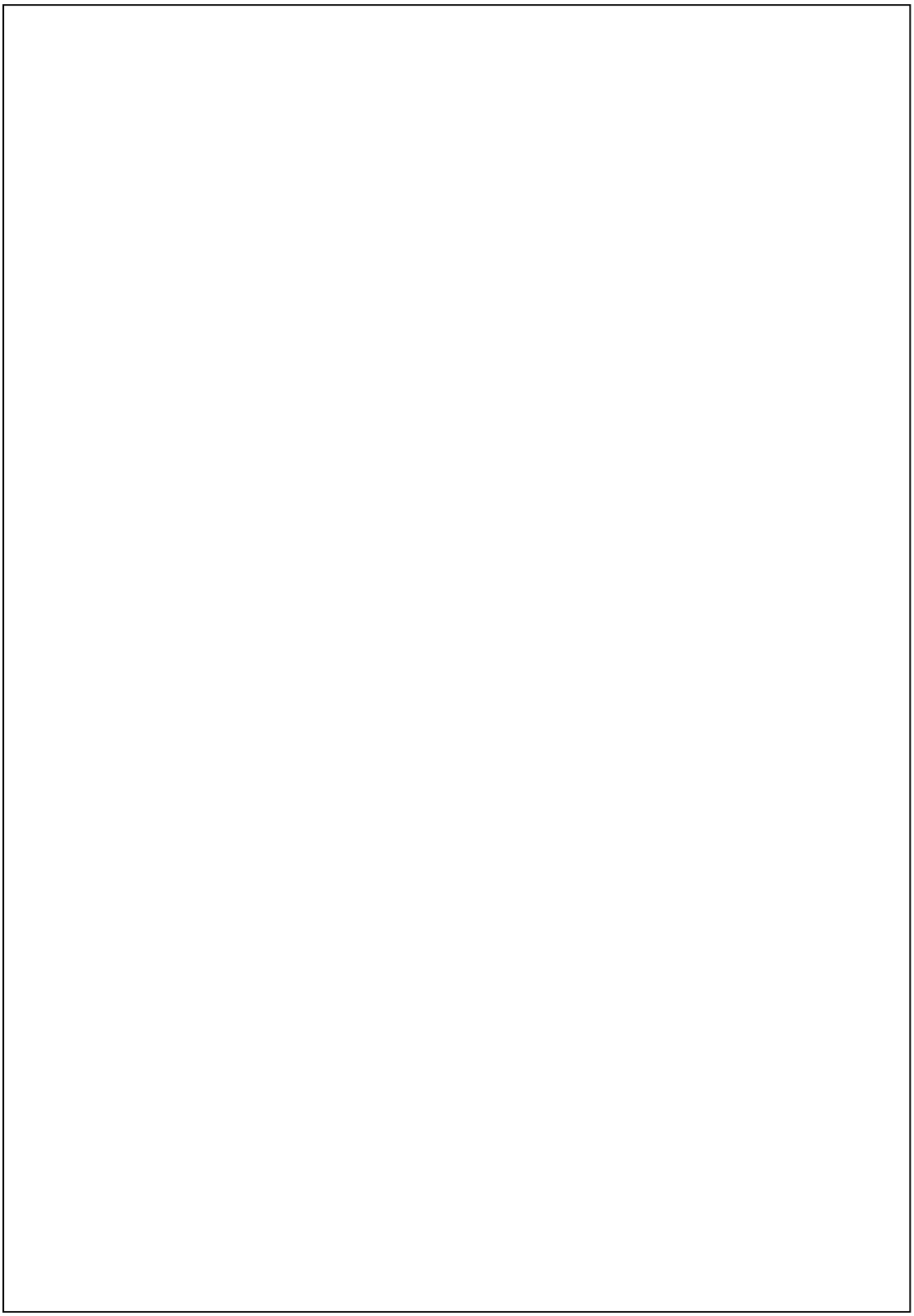
```
>>> sayi1 = Tamsayi4islemYeni(8)
>>> sayi2 = Tamsayi4islemYeni(3)
>>> print sayi1
8
>>> print sayi2
3
>>> arti = sayi1 + sayi2
>>> print arti
83
>>> eksi = sayi1 - sayi2
>>> print eksi
38
>>> carpi = sayi1 * sayi2
>>> print carpi
512
>>> bolu = sayi1 / sayi2
>>> print bolu
2.0
```

İpucu

"Operatör Gölgeleme" bölümünü inceleyiniz.

Sonuç

Gerçekleştiriminizi ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:



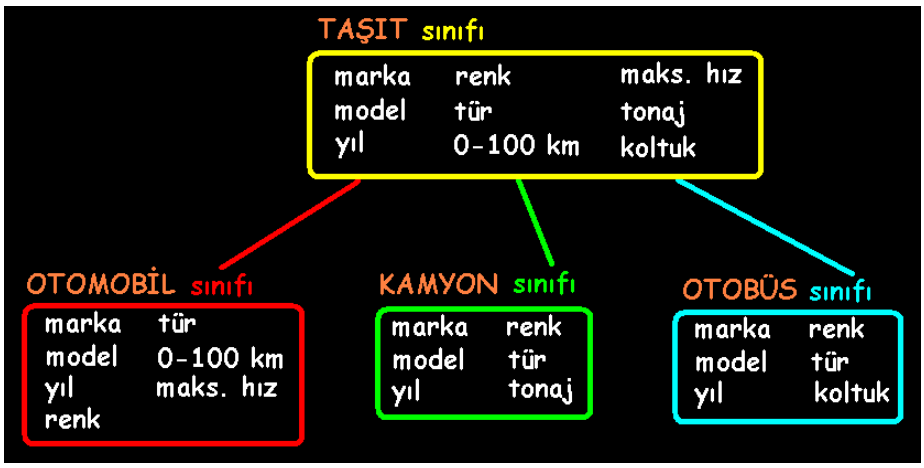
LABORATUVAR ÇALIŞMASI – 9

Bu Çalışmanın Amacı

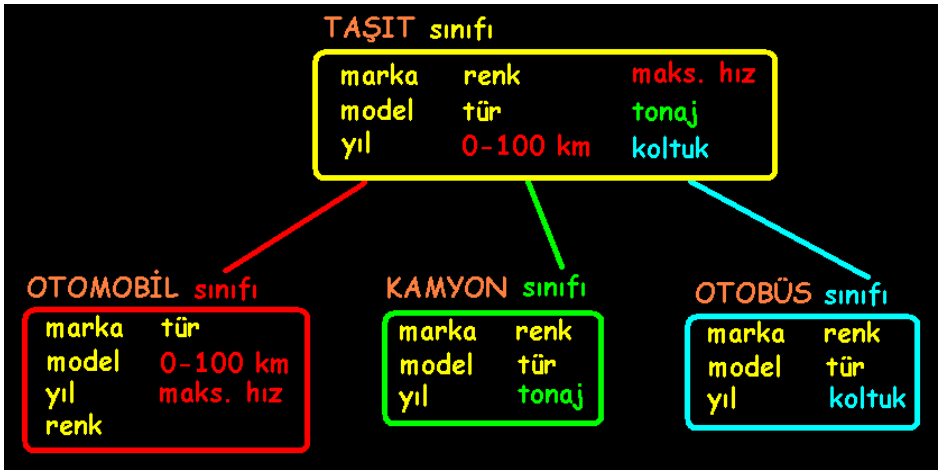
Bu çalışmadaki amacımız, kalıtım kavramı ve grafiksel kullanıcı arayüzü kullanımı hakkında öğrendiklerimizi pekiştirmektir.

Kalıtım

Programlamada bir sınıfın, başka bir sınıfın özelliklerinden yararlanmasına, dolayısı ile de onunla ortak özellikler bulundurmasına "**kalıtım**" ismi verilir. Bu bölümde, "kalıtım" kavramını anlatmak için "Tasit" isimli bir sınıftan yararlanılacaktır. Bu sınıf; otomobil, kamyon ve otobüs türünden olan taşıtların bilgilerinin tutulduğu "Otomobil", "Kamyon" ve "Otobus" isimli üç sınıfla beraber yer almaktadır. Bu üç sınıf, "Tasit" sinifindan kalıtlamaktadırlar:



Bu üç sınıftan her biri "Taşıt" sınıfının pek çok özelliğine ("özellik" den kasıt: sınıf nesnesinin alt alanları) sahip olmakla birlikte; "Otomobil" sınıfı için 'tonaj' ve 'koltuk' bilgilerine, "Kamyon" sınıfı için '0-100 km', 'maks. hız' ve 'koltuk' bilgilerine, "Otobüs" sınıfı içinse '0-100 km', 'maks. hız' ve 'tonaj' bilgilerine gerek yoktur. Altta yer alan üç sınıftan her biri, 'marka', 'model', 'yıl', 'renk' ve 'tür' özelliklerini ortak olarak aldıktan sonra sadece kendileri için gerekli olan özellikleri alacaklar, diğerlerini kullanmayacaklardır:



Yukarıdaki şekillerde ifade edilen sınıfların yer aldığı betik dosyası aşağıdadır (Sınıf isimlerinde Türkçe karakter kullanılmadığı için "Taşıt" sınıfı "Tasit" olarak, "Otobüs" sınıfı ise "Otobus" olarak geçmektedir. Betik dosyasında ise çok uzun satırlar parçalanarak 2. ve sonraki parçalar ilk parçanın altına sırayla ve sağa dayalı olarak yazılmıştır.):

```
class Tasit(object):
    """Taşıt nesnelerinin (Otomobil, Kamyon, Otobüs) yer
                                          aldığı sınıftır."""

    def __init__(self, marka='?', model='?', yil=0,
            renk='?', tur='taşıt', sifirYuzKm=0, maksHiz=0,
                                        tonaj = 0, koltuk=0):
        self.marka = marka
        self.model = model
        self.yil = yil
        self.renk = renk
        self.tur = tur
        self.sifirYuzKm = sifirYuzKm
        self.maksHiz = maksHiz
        self.tonaj = tonaj
        self.koltuk = koltuk

    def __str__(self):
        return 'Türü      : %s\nMarka     : %s\nModel     :
            %s\nYıl       : %d\nRenk      : %s\n' % (self.tur,
                self.marka, self.model, self.yil, self.renk)

    def OnemliBilgiler(self):
        print '%d üretimi %s %s' % (self.yil, self.marka,
                                                self.model)


#"Tasit" sınıfından kalıtlayan "Otomobil" sınıfı
class Otomobil(Tasit):
    """Otomobil bilgilerinin yer aldığı sınıftır."""

    def __init__(self, marka='?', model='?', yil=0,
                            renk='?', sifirYuzKm=0, maksHiz=0):
        Tasit.__init__(self, marka, model, yil, renk,
                    'otomobil', sifirYuzKm, maksHiz, -1, -1)
        #Otomobil sınıfında "tonaj" bilgisini tutmaya lüzum
                                                        yok.
        del self.tonaj
        del self.koltuk

    def __str__(self):
        return 'Türü      : %s\nMarka     : %s\nModel     :
                            %s\nYıl       : %d\nRenk      : %s\n
                    0-100 km : %g sn\nMaks. Hız: %d km/h\n'
                % (self.tur, self.marka, self.model, self.yil,
                        self.renk, self.sifirYuzKm, self.maksHiz)

    def OnemliBilgiler(self):
        Tasit.OnemliBilgiler(self)
        print 'Tür : %s\nMaks. Hız : %d km/h' % (self.tur,
                                                self.maksHiz)
```

>>devam>>

```
>>devam>>

class Kamyon(Tasit):
    """Kamyon bilgilerinin yer aldığı sınıftır."""

    def __init__(self, marka='?', model='?', yil=0,
                                              renk='?', tonaj=0):
        Tasit.__init__(self, marka, model, yil, renk,
                                    'kamyon', -1, -1, tonaj, -1)
        del self.sifirYuzKm
        del self.maksHiz
        del self.koltuk

    def __str__(self):
        return 'Türü      : %s\nMarka     : %s\nModel     :
         %s\nYıl       : %d\nRenk      : %s\nTonaj     : %g ton'
              % (self.tur, self.marka, self.model, self.yil,
                                        self.renk, self.tonaj)

    def OnemliBilgiler(self):
        Tasit.OnemliBilgiler(self)
        print 'Tür : %s\nYük Kapasitesi : %g ton\n'
                                       % (self.tur, self.tonaj)


class Otobus(Tasit):
    """Otobüs bilgilerinin yer aldığı sınıftır."""

    def __init__(self, marka='?', model='?', yil=0,
                                            renk='?', koltuk=0):
        Tasit.__init__(self, marka, model, yil, renk,
                                  'otobüs', -1, -1, -1, koltuk)
        del self.sifirYuzKm
        del self.maksHiz
        del self.tonaj

    def __str__(self):
        return 'Türü      : %s\nMarka     : %s\nModel     :
       %s\nYıl       : %d\nRenk      : %s\nKoltuk    : %d adet'
              % (self.tur, self.marka, self.model, self.yil,
                                        self.renk, self.koltuk)

    def OnemliBilgiler(self):
        Tasit.OnemliBilgiler(self)
        print 'Tür : %s\nYolcu Kapasitesi : %d kişi\n'
                                       % (self.tur, self.koltuk)
```

"Tasit" sınıfının "__init__" ve "__str__" fonksiyonlarını, daha önceden "LABORATUVAR ÇALIŞMASI - 8" içerisinde anlatıldığı gibi tanımladık. Bir de, "OnemliBilgiler" isminde, ekrana taşıtın marka, model ve üretim yılı bilgilerini yazmaya yarayan bir fonksiyon tanımladık. "Otomobil" sınıfını tanımlarken ise, "`class Otomobil(Tasit):`" demekle biz

"'Otomobil' isimli sınıf 'Tasit' isimli sınıftan kalıtlar yani 'Tasit' sınıfının içerisinde yer alan özelliklerden bir kısmını ya da tamamını kullanabilir." demiş oluyoruz. "Otomobil" sınıfının "init" fonksiyonuna baktığımızda, sil baştan yeni bir fonksiyon tanımlaması yapmak yerine "Tasit" sınıfının "init" fonksiyonunun çağrıldığını görürüz. "Otomobil" sınıfında 'tonaj' ve 'koltuk' bilgilerini tutmaya gerek olmadığı için "**del**" komutu kullanılarak bunlar sınıftan çıkarılmıştır ("**Tasit.__init__()**" fonksiyonunda da 'tonaj' ve 'koltuk' parametrelerine karşılık gelen değerlere herhangi bir sayı girilebilir, örnekte '-1' girilmiştir.). "Otomobil" sınıfındaki "**__str__()**" fonksiyonu sıradan bir tanımlamaya sahipken "OnemliBilgiler" fonksiyonunun tanımlanması esnasında da "Tasit" sınıfının "OnemliBilgiler" fonksiyonundan yararlanılmış, üzerine eklemeler yapılmıştır. **Sonuç olarak, "Tasit" sınıfından kalıtlayan "Otomobil" sınıfı elde edilmiştir. "Otomobil" sınıfı, "Tasit" sınıfının pek çok özelliğini bünyesinde barındırmaktadır. Bu işlemler esnasında "Tasit" sınıfında yer alan kodlar da yeniden kullanılmıştır.**

"Kamyon" ve "Otobus" isimli sınıflar da gerçekleştirim mantığı bakımından "Otomobil" sınıfı ile aynı özelliklere sahiptirler.

Şimdi ise bu sınıfların kullanım örneğini inceleyelim (sınıfların yer aldığı betik dosyasından F5 tuşu ile Python Shell ekranına geçerek):

```
>>> tst = Tasit()
>>> print tst
Türü      : taşıt
Marka     : ?
Model     : ?
Yıl       : 0
Renk      : ?

>>> tst.OnemliBilgiler()
0 üretimi ? ?
>>> otm1 = Otomobil()
>>> print otm1
Türü      : otomobil
Marka     : ?
Model     : ?
Yıl       : 0
Renk      : ?
0-100 km  : 0 sn
Maks. Hız: 0 km/h

>>> otm2 = Otomobil('Volvo', 's80 T6', 2003, 'beyaz', 8.8, 250)
>>> print otm2
Türü      : otomobil
Marka     : Volvo
Model     : s80 T6
Yıl       : 2003
Renk      : beyaz
0-100 km  : 8.8 sn
Maks. Hız: 250 km/h

>>> otm2.OnemliBilgiler()
2003 üretimi Volvo s80 T6
Tür : otomobil
Maks. Hız : 250 km/h
>>> kmy = Kamyon('Bedford', 'Genoto', 1978, 'mavi', 9.5)
>>> print kmy
Türü      : kamyon
Marka     : Bedford
Model     : Genoto
Yıl       : 1978
Renk      : mavi
Tonaj     : 9.5 ton
>>> kmy.OnemliBilgiler()
1978 üretimi Bedford Genoto
Tür : kamyon
Yük Kapasitesi : 9.5 ton

>>> otb = Otobus('Mitsubishi', 'Prenses', 1991, 'yeşil', 46)
>>> print otb
Türü      : otobüs
Marka     : Mitsubishi
Model     : Prenses
Yıl       : 1991
Renk      : yeşil
Koltuk    : 46 adet
>>> otb.OnemliBilgiler()
1991 üretimi Mitsubishi Prenses
Tür : otobüs
Yolcu Kapasitesi : 46 kişi
```

Alıştırmalar

Alıştırma – 1

Görev

"Tasit" sınıfının içerisinde, bu sınıftan kalıtlayan "Traktor" isminde bir sınıf oluşturunuz. "Traktor" sınıfında özellik olarak 'marka', 'model', 'yıl', 'tür' ve 'tonaj' bulunmalıdır. Bu sınıfın "__str__()" ve "OnemliBilgiler" fonksiyonları için, "Kamyon" sınıfındaki "__str__()" ve "OnemliBilgiler" fonksiyonlarını inceleyiniz.

Sonuç

Yeni oluşturduğunuz sınıfı ve / veya karşılaştığınız problemleri aşağıdaki kutunun içerisine yazınız:

Alıştırma – 2

Görev

Alıştırma - 1' de gerçekleştirmeniz beklenen "Traktor" sınıfına, ek bir özellik olarak 'boyut' ekleyiniz (traktörlerin 'küçük', 'orta', 'büyük' gibi boyut bilgilerini tutmak için). Bu işlem için diğer sınıfların tümünü etkileyecek bir değişiklik yapmanız gerektiğini ve gerekli değişiklikleri yaptıktan sonra tüm sınıfların doğru bir biçimde çalışması gerektiğini unutmayınız. 'boyut' özelliğini "__str__()" ve "OnemliBilgiler" fonksiyonlarına eklemeyi unutmayınız.

Sonuç

Tüm sınıfların en son hallerini ve / veya karşılaştığınız problemleri aşağıdaki kutuların içerisine yazınız: