tuple'lar ve değişebilirlik

→ bileşik yapılar: dizgi, liste

→ karakter / herhangi bir şey

→ değiştirilebilirlik/değiştirilemezlik

→ tuple: liste gibi herhangi bir şey barındırır

→ dizgi gibi değiştirilemezdir

→ virgüller ayrılmış değerler

```
>>> tup = 2, 4, 6, 8, 10
```

→ gerekmese de gelenekselleşmiş gösterim,

```
>>> tup = (2, 4, 6, 8, 10)
```

tuple

→ tek öğeli tuple için sona bir vürgül gerekir

```
>>> tup = (5, )
>>> type(tup)
<type 'tuple'>
```

→ virgül unutulduğunda

```
>>> tup = (5)
>>> type(tup)
<type 'int'>
```

liste gibi davranış

→ indisleme

```
>>> tup = ('a', 'b', 'c', 'd', 'e')
>>> tup[0]
'a'
```

→ dilimleme

```
>>> tup[1:3]
('b', 'c')
```

dizgi gibi davranış

→ değiştirmeye çalışma

```
>>> tup[0] = 'X'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

→ değiştirmeniz gerekiyorsa, dilimle al, ekle, aynı değişkene ata

```
>>> tup = ('X', ) + tup[1:]
>>> tup
('X', 'b', 'c', 'd', 'e')
```

→ veya listeye çevir, işle, tuple'a geri dön

```
>>> tup
('X', 'b', 'c', 'd', 'e')
>>> tup = list(tup)
>>> tup
>>> tup[0] = 'a'
>>> tup = tuple(tup)
>>> tup
('a', 'b', 'c', 'd', 'e')
```

tuple ataması

→ geleneksel yöntem

```
>>> a = 3; b = 5
>>> gecici = a
>>> a = b
>>> b = gecici
>>> a, b
(5, 3)
```

→ tuple atamalı yöntem

```
>>> a = 3; b = 5
>>> a, b = b, a
>>> a, b
(5, 3)
```

→ sol ve sağ taraftaki değer sayısı aynı olmalıdır

```
>>> a, b, c, d = 1, 2, 3
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ValueError: need more than 3 values to unpack
```

geri dönüş değeri olarak tuple

→ swap

```
def swap(x, y):
    return y, x
```

→ böyle çağır: a, b = swap(a, b)

→ değişken ömürleri

```
def swap(x, y): # yanlis surum
    x, y = y, x
```

→ neden, yanlış?

saf (pure) fonksiyonlar

→ PB:6374

```
1   import doctest
2
3   def insert_in_middle(val, lst):
4       middle = len(lst) / 2
5       lst[middle:middle] = [val]
6
7   def make_empty(seq):
8       """
9         >>> make_empty([1, 2, 3, 4])
10        []
11        >>> make_empty(('a', 'b', 'c'))
12        ()
13        >>> make_empty("No, not me!")
14        ''
15      """
16
17  def insert_at_end(val, seq):
18      """
19        >>> insert_at_end(5, [1, 3, 4, 6])
20        [1, 3, 4, 6, 5]
21        >>> insert_at_end('x', 'abc')
22        'abcx'
23        >>> insert_at_end(5, (1, 3, 4, 6))
24        (1, 3, 4, 6, 5)
```

test

→ test

```
>>> from d11_seqtools import *
>>> insert_in_middle('c', my_list)
>>> my_list
['a', 'b', 'c', 'd', 'e']
>>> from d11_seqtools import *
>>> my_list = ['a', 'b', 'd', 'e']
>>> insert_in_middle('c', my_list)
>>> my_list
['a', 'b', 'c', 'd', 'e']
```

tuple

→ tuple'la çalışırken

```
>>> my_tuple = ('a', 'b', 'd', 'e')
>>> insert_in_middle('c', my_tuple)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "./d11_seqtools.py", line 3, in insert_in_middle
TypeError: 'tuple' object does not support item assignment
```

→ onaralım

```
1       def insert_in_middle(val, tup):
2           middle = len(tup) / 2
3           return tup[:middle] + (val, ) + tup[middle:]
```

→ tuple düzeldi, liste aksıyor

hepsine uyacak bir işlev

→ hepsine uyacak bir işlev

```
1       def encapsulate(val, seq):
2           if type(seq) == type(""):
3               return str(val)
4           if type(seq) == type([]):
5               return [val]
6           return (val, )
7
8       def insert_in_middle(val, seq):
9           middle = len(seq) / 2
10          return seq[:middle] + encapsulate(val, seq) + seq[middle:]
```

test

→ test

```
>>> from d11_seqtools_enc import *
>>> my_string = "abde"
>>> my_list   = ['a', 'b', 'd', 'e']
>>> my_tuple  = ('a', 'b', 'd', 'e')
>>> insert_in_middle('c', my_string)
'abcde'
>>> insert_in_middle('c', my_list)
['a', 'b', 'c', 'd', 'e']
>>> insert_in_middle('c', my_tuple)
('a', 'b', 'c', 'd', 'e')
```

değiştirilebilirlik

→ dizgi/tuple'a değiştirilebilirlik kazandırma

```
>>> my_string = "abde"
>>> my_string = insert_in_middle('c', my_string)
>>> my_string
'abcde'
```

özyineli

veri yapısı: verinin düzenleniş biçimi

→ oy sayımı yapacağız

→ oylar şehirlerden, şehir oyları ise ilçelerden vs

→ oy listesi (sayı listesi) herhangi bir öğe içerebilir

→ *sayılar*

→ *içiçe sayı listesi*

→ özyineli bir tarif verdik

→ özyineli veri yapısı

oy sayımı

→ python'un yerleşik sum işlevi (PB:6376)

```
>>> sum([1, 2, 8])
11
>>> sum((1, 2, 8))
11
```

→ fakat içiçe sayı listesinde çöker

```
>>> sum([1, 2, [11, 13], 8])
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int'
 and 'list'
>>>
```

özyineleme

→ içiçe sayı listesini idare etmek için

→ toplama işlevinin her bir iç liste için çağrılması gerekir

→ çağırıldığı yer (toplama işlevi), çağrılan işlev (toplama işlevi)

→ özyineli çağrı

özyineli gerçekleme

→ gerçekleme

```
1       def recursive_sum(nested_num_list):
2           sum = 0
3           for elem in nested_num_list:
4               if type(elem) == type([]):
5                   sum = sum + recursive_sum(elem)
6               else:
7                   sum = sum + elem
8           return sum
```

→ test

```
1   >>> from d11_recursive_sum import *
2   >>> recursive_sum([1, 2, [11, 13], 8])
3   35
```

özyineli çağrı

→ en büyüğü bulma

```
1       def recursive_max(nested_num_list):
2           """
3             >>> recursive_max([2, 9, [1, 13], 8, 6])
4             13
5             >>> recursive_max([2, [[100, 7], 90], [1, 13], 8, 6])
6             100
7             >>> recursive_max([2, [[13, 7], 90], [1, 100], 8, 6])
8             100
9             >>> recursive_max([[[13, 7], 90], 2, [1, 100], 8, 6])
10            100
11          """
12          largest = nested_num_list[0]
13          while type(largest) == type([]):
14              largest = largest[0]
15          for element in nested_num_list:
16              if type(element) == type([]):
17                  max_of_elem = recursive_max(element)
18                  if largest < max_of_elem:
19                      largest = max_of_elem
20              else:                              # element is not a list
21                  if largest < element:
22                      largest = element
23          return largest
```

sonsuz

→ sonduz yinelem

```
#
# infinite_recursion.py
#
def recursion_depth(number):
    print "Recursion depth number %d." % number
    recursion_depth(number + 1)

recursion_depth(0)
```

→ sonuç

```
$ python infinite_recursion.py
...
  File "infinite_recursion.py", line 3, in recursion_depth
    recursion_depth(number + 1)
RuntimeError: maximum recursion depth exceeded
```

istisnalar

- → çalışma zamanı hatası -> **istisna** ortaya çıkar
- → program durur ve hata dökümünü verir
- → bu döküm ortaya çıkan istisnayla biter

```
>>> print 55 / 0
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

örnekler

→ var olmayan bir liste öğesine erişmeye çalışmak

```
>>> a = []
>>> print a[55]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
IndexError: list index out of range
```

→ tupleda öğe ataması yapmaya çalışmak:

```
>>> tup = ('a', 'b', 'd', 'e')
>>> tup[2] = 'c'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

özet

- → hata türü: ZeroDivisionError, IndexError, TypeError
- → hata ayrıntıları
- → http://docs.python.org/library/exceptions.html#bltin-exceptions

istisna idaresi

→ istisna ortaya çıktığında program sonlanmasın

→ istisna idaresi: try ... except

```
filename = raw_input('Enter a file name: ')
try:
    f = open (filename, "r")
except:
    print 'There is no file named', filename
```

→ böylesi bir durumda programın çökmesinin hiç gereği yoktur

exists

→ böyle de kullanabilirsiniz

```
def exists(filename):
    try:
        f = open(filename)
        f.close()
        return True
    except:
        return False
```

profesyonelce

→ pratik örnek

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as (errno, strerror):
    print "I/O error({0}): {1}".format(errno, strerror)
except ValueError:
    print "Could not convert data to an integer."
except:
    print "Unexpected error:", sys.exc_info()[0]
    raise
```

istisna tetikleme

→ siz de bir istisna üretebilirsiniz

```
#
# learn_exceptions.py
#
def get_age():
    age = input('Please enter your age: ')
    if age < 0:
        raise ValueError, '%s is not a valid age' % age
    return age
```

test

→ test

```
>>> get_age()
Please enter your age: 42
42
>>> get_age()
Please enter your age: -2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "learn_exceptions.py", line 4, in get_age
    raise ValueError, '%s is not a valid age' % age
ValueError: -2 is not a valid age
>>>
```

→ bir yerlerde bunu idare edebilirsiniz, nasıl?

istisna: sonsuz döngü

→ önceki özyineli sonsuz döngüyü hatırlayın

```
#
# infinite_recursion.py
#
def recursion_depth(number):
    print "Recursion depth number %d." % number
    try:
        recursion_depth(number + 1)
    except:
        print "Maximum recursion depth exceeded."

recursion_depth(0)
```

→ artık program çökmeyecek!

kuyruk çağrı: faktoriyel

iki tanım

1. n! = n*(n-1)*...*2*1
2. n! = n * (n - 1)!

→ ikincisini nasıl kodlarız?

```
def faktoriyel(n):
    if n == 0:
        return 1
    else:
        return n * faktoryel(n-1)
```

kuyruk çağrı: fibonacci

→ fib(0) = fib(1) = 1

→ fib(n) = fib(n-1) + fib(n-2)

```
def fibonacci (n):
    if n == 0 or n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

liste kavraması

→ Liste kavraması kısa, matematiksel bir sözdizimi kullanarak

→ diğer listelerden yeni listeler yaratmaya yarayan

→ sözdizimsel bir yapıdır

```
>>> sayilar = [1, 2, 3, 4]
>>> [x**2 for x in sayilar]
[1, 4, 9, 16]
>>> [x**2 for x in sayilar if x**2 > 8]
[9, 16]
>>> [(x, x**2, x**3) for x in sayilar]
[(1, 1, 1), (2, 4, 8), (3, 9, 27), (4, 16, 64)]
>>> dosyalar = ['bin', 'Data', 'Desktop', '.bashrc', '.ssh', '.vimrc']
>>> [isim for isim in dosyalar if isim[0] != '.']
['bin', 'Data', 'Desktop']
>>> harfler = ['a', 'b', 'c']
>>> [n*harf for n in sayilar for harf in harfler]
['a', 'b', 'c', 'aa', 'bb', 'cc', 'aaa', 'bbb', 'ccc', 'aaaa', 'bbbb', 'cccc']
>>>
```

genelleştirilmiş

→ genelleştirilmiş hali

```
[expr for item1 in seq1 for item2 in seq2 ...
 for itemx in seqx if condition]
```

→ açık hali

```
output_sequence = []
for item1 in seq1:
    for item2 in seq2:
        ...
            for itemx in seqx:
                if condition:
                    output_sequence.append(expr)
```

örnek

→ tree : d11_tree.py

alıştırmalar

1.

```
1  def swap(x, y):       # yanlis surum
2       print  "before swap statement: id(x):", id(x), "id(y):", id(y)
3       x, y = y, x
4       print  "after swap statement: id(x):", id(x), "id(y):", id(y)
5
6  a, b = 0, 1
7  print  "before swap function call: id(a):", id(a), "id(b):", id(b)
8  swap(a, b)
9  print  "after swap function call: id(a):", id(a), "id(b):", id(b)
```

2. d11_seqtools.py

3. d11_recursive

alıştırmalar

4.

```
>>> num = readposint()
Please enter a positive integer: yes
yes is not a positive integer.  Try again.
Please enter a positive integer: 3.14
3.14 is not a positive integer.  Try again.
Please enter a positive integer: -6
-6 is not a positive integer.  Try again.
Please enter a positive integer: 42
>>> num
42
>>> num2 = readposint("Now enter another one: ")
Now enter another one: 31
>>> num2
31
>>>
```

5. çıktı? (8)

alıştırmalar

6. faktoriyel işlevinin yineli versiyonunu yazın. süre karşılaştırın
7. litter.py, o anki ve alt dizinlerde trash.txt oluştursun. cleanup.py ise trash.txt leri temizlesin