Python:
Функции и структуры данных

Стеки

- Стек (LIFO) – это список
- Добавление элемента в стек: append(x)
- Извлечение элемента с вершины стека: pop()

```
>>> stack = [‘+’, ‘-’, ‘+’, ‘*’]
>>> stack.append(‘-’)
>>> stack.append(‘/’)
>>> stack
[‘+’, ‘-’, ‘+’, ‘*’, ‘-’, ‘/’]
>>> stack.pop()
‘/’
>>> stack
[‘+’, ‘-’, ‘+’, ‘*’, ‘-’]
```

Очереди

- Очередь (FIFO) – это тоже список
- Добавление элемента в очередь: append(x)
- Извлечение элемента «в порядке очереди»: pop(0)

```
>>> queue = [“repka”, “dedka”, “babka”, “vnuchka”]
>>> queue.append(“zhuchka”)
 >>> queue
[‘repka’, ‘dedka’, ‘babka’, ‘vnuchka’, ‘zhuchka’]
>>> queue.pop(0)
‘repka’
>>> queue.pop(0)
‘dedka’
>>> queue
[‘babka’, ‘vnuchka’, ‘zhuchka’]
```

Словари

- Ассоциативный массив (как хэш в Perl)
- Неупорядоченное множество пар ключ: значение
- Ключи – любого типа, не допускающего изменений (строки, числа, кортежи, не включающие в себя списки)
- Значения – любого типа

```
>>> dict = {}                                  # пустой словарь
>>> circus = {“lion” : 4, “hippo” : 1, “giraffe” : 2}
>>> circus[“hippo”]
1
>>> circus[“snake”] = 7                        # добавление ключа
>>> circus[“lion”] = 5                         # изменение ключа
>>> circus
{‘hippo’: 1, ‘lion’: 5, ‘giraffe’: 2, ‘snake’: 7}
```

Словари

```
>>> len(circus)                          # количество элементов в словаре
5
>>> circus[“cat”] = “yes, please!”       # разные типы значений
>>> circus[42] = “number”                # разные типы ключей
>>> del circus[“hippo”]                  # удаление ключа
>>> circus
{42: ‘number’, ‘cat’: ‘yes, please!’, ‘lion’: 5, ‘giraffe’: 2, ‘snake’: 7}
>>> circus.keys()                        # возвращает список ключей
[42, ‘cat’, ‘lion’, ‘giraffe’, ‘snake’]
>>> circus.values()                      # возвращает список значений
[‘number’, ‘yes, please!’, 5, 2, 7]
>>> ‘dog’ in circus                      # проверяет наличие ключа в словаре
False
```

- При обращении по несуществующему ключу – исключение KeyError

Лирическое отступление: Исключения

>>> print circus[‘dog’]

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>

print circus['dog']

KeyError: 'dog‘

Можно поймать это исключение:

```
try:
    print circus['dog']
except KeyError:
    print "No such key in the dictionary"
```

Сперва выполняется ветвь **try**, если все хорошо - ветвь **except** пропускается. Если генерируется исключение и его тип совпадает с указанным, выполняется ветвь **except**.

Словари

Как отсортировать словарь по ключам:

1)

```
keys = circus.keys()
keys.sort()
print [circus[k] for k in keys]
```

2)

```
circitems = circus.items()
```

метод items() возвращает список пар (ключ, значение)

```
circitems.sort()
print [value for key, value in circitems]
```

Функции

```
def catalan(n):
   ``` This function computes the n-th Catalan
   number, assuming that C(0) = 1. '''
   def binCoef(b, a):
       if b < a:
             return 0
       if a == 0:
             return 1
       return (b-a+1)*binCoef(b, a-1)/a
   return binCoef(2*n, n)/(n+1)
```

Ключевое слово для определения функции: **def**

Функции всегда возвращают значение, хотя бы None.
```

Функции

Функции – тоже объекты.

```
c = catalan                    # их можно присваивать
c(10)                          # то же самое, что catalan(10)
def aprogr(a, d):
   def nmember(n):
       return a + (n-1)*d
   return nmember              # их можно возвращать из функции

progr1 = aprogr(0, 3)
print progr1(4)                # четвертый член последовательности
```

Результат:

9

Функции

Передача аргументов в функцию - по ссылке.

```
def dostuff(mylist) :
   """ Appends [1, 2, 3] to the list. """
   mylist.append([1, 2, 3])
   mylist = ['Katya', 'Kolya', 'Vitya']
   mylist.append('Sveta')
   print mylist

a = [4, 5, "f"]
dostuff(a)          # результат: ['Katya', 'Kolya', 'Vitya', 'Sveta']
print a             # результат: [4, 5, 'f', [1, 2, 3]]
```

Функции

```
def square(n) :
   n *= n
a = 3
square(a)
print a
```

Результат выполнения:

3

Потому что число – простой тип, оно передается по значению.

Внимание, вопрос: как передаются строчки, и можно ли изменить строчку внутри функции?

Функции

Переменные внутри функции – локальные. Поиск переменных: сперва среди локальных, потом среди глобальных, потом среди встроенных.

```
n = 10
def printn():
  print n                  # переменная n видна внутри функции
def changeandprintn()
  n = 2                    # теперь n – это локальная переменная
  print n
printn()
changeandprintn()
print n
```

Результат выполнения:

10

2

10

Функции

```
n = 10
def reallychangeandprintn()
   global n          # переменная n – глобальная
   n = 2
   print n
reallychangeandprintn()
print n
```

Результат:

2

2

global a, b, c - указывает, что идентификаторы a, b, c в текущем блоке ссылаются на глобальные переменные

Функции

Значения по умолчанию:

```
def f(name, age = 23, occupation = 'student'):
   print 'Name:', name
   print 'Age:', age
   print 'Occupation:', occupation
```

Можно вызвать как:

f('Arthur Lee Allen', 45, 'homicidal maniac')

f('Nina')

f(occupation = 'PHD student', age = 24, name = 'Igor')

f('Lisa', occupation = 'teacher')

Функции

Замечания:

1. значения по умолчанию вычисляются в месте определения функции

```
n = 10
def getAccess(age = n):
    if (age < 18) :
        print “Access denied.”
        website = ‘http://www.webiki.ru’
    else :
        print “Welcome!”
        website = ‘http://www.reddit.com’
n = 20
getAccess()          # результат выполнения: Access denied.
```

Функции

Замечания:

2. значения по умолчанию вычисляются один раз

```
def f(a, list = [])
    list.append(a)
    print list
f('Nikita')          # результат выполнения: ['Nikita']
f('Andrey')          # результат выполнения: ['Nikita', 'Andrey']
f('Yura')            # результат выполнения: ['Nikita', 'Andrey', 'Yura']
```

Функции

Как передать в функцию произвольное число аргументов:

f([formal_args,] *tuple): tuple – кортеж, содержащий аргументы, не входящие в список формальных параметров

```
def mean(*args):
   sum = 0.
   for a in args:
       sum += a
   return sum/len(args)

print mean(1, 2, 3, 4, 5)      # результат: 3
print mean(40, 3)              # результат: 21.5
```

Функции

Еще веселее:

f([formal_args,] *tuple, **dict): dict – словарь, содержащий именованные аргументы, не входящие в список формальных параметров

```
def f(*args, **words):
   for a in args:
       print a
   for k in words.keys():
       print k, ':', words[k]

f('answer', 42, monkeys=12, circus='one and only')
```

Результат выполнения:

answer

42

monkeys : 12

circus : one and only