Лекция 3
Разработка программ на Python

Давыдов Виталий Валерьевич (ГАИШ МГУ, Постгрес-ПРО)

Повторение пройденного

- **Последовательности**
  - Списки (lists, mutable)
  - Кортежы (tuples, immutable)
  - Строки (immutable)
  - Срезы (slices)
  - Копирование и глубокое копирование (deep copy)
- **Словари**
  - Стандартные словари (dict)
  - Хеширование (хеш-функция, хеш-таблицам коллизии)
  - Ключ словаря – immutable, hashable объект
- **Виды операции присваивания**
- **Операция сравнения**

Повторение - Вопросы

- `L = [9,8,7,6,5]; L[True], L[False]`
- `L = [1,2,3,4,5]; L[[True, False, False, True, True]]`
- `D = { [1,2]: 1, [3,4]: 2 }`
- `D.update(D)`
- Можно ли создать тапл из одного элемента?
- `[1, 2, 3, (4)]`
- `[1, 2, 3, (4,)]`

Форматирование кода

```
import time
import datetime

def generate(step, duration, azbeg, azstep, altbeg, altstep):
    tmsbeg = int(1000 * datetime.datetime.now().timestamp())
    tmsend = int(tmsbeg) + duration;
    tms, az, alt = tmsbeg, azbeg, altbeg
    while tms < tmsend:
        yield tms, az, alt
        tms, az, alt = tms + step, az + azstep, alt + altstep

for tms, az, alt in generate(100, 300000, 100.0, 0.01, 45.0, 0.01):
    print(tms, az, alt)
```

строка заголовка:
.... вложенный блок операторов

- Код разделен на блоки
- Границы блоков опред. автоматически
- Конец строки – конец оператора
- Конец отступа – конец блока
- Можно располагать операторы на одной строке (разделять через ;)
- Можно разнести однострочный оператор на множество строк, используя (), [], {}.
- Выражения могут быть многострочными, используя ().
- Один оператор можно располагать на той же строке после двоеточия.

Операторы

- **Проверки if**
  - if, elif, else
  - Допускают множественное ветвление
  - Могут быть вложенными
  - Логические операторы в выражениях (or, and)
- **Тернарное выражение if**
  - a = y if x else z
- **Цикл while**
  - while, else
- **Цикл for**
  - for in, else
- **Операторы циклов**
  - pass
  - continue
  - break

Цикл for (примеры)

```
for x in [1, 2, 3, 4]:
     print(x)

for x in 1, 2, 3, 4, 5:
     print(x)

for x in (1, 2, 3, 4, 5):
     print(x)

for x in “Hello“:
    print(x)

for x in range(100):
     print(x)

for key in D:
     print(key)
```

```
T = [(1,2), (3,4), (5,6)]
for (a, b) in T:
     print(a, b)

for tup in T:
     a, b = tup

for (k, v) in D.items():
     print(k, v)

L = [((1,2),3), ((4,5),6)]
for ((a,b),c) in T:
     print(a, b, c)

L = [(1,2,3,4), (5,6,7,8)]
for (a,*b,c) in L:
     print(a,b,c,d)
```

```
for key in tests:
     for item in items:
          if item == key:
               print(key)

for x in L:
     if x == 2:
          break
else:
     print(„not found“)

L1 = […]
L2 = […]
for (x, y) in zip(L1, L2):
     print(x, y)
```

Функции

- Используются для структурирования кода
- Могут принимать входные и возвращать выходные значения
- Есть несколько типов функций:
  - Простые функции (def + return)
  - Функции-генераторы (def + yield)
  - Лямба-функции (lambda, безымянные)
  - Декораторы (@)
- Функции — это объекты в Python
  - id, type
  - Присваивать переменным
  - Хранить в структурах данных
  - Передавать в качестве аргументов другим функциям
  - Возвращать в качестве значений из других функций
- Функции могут быть вложенными
- Могут захватывать состояние (замыкание, closure)

Типы функций

```
def myfunc():
    print(a)

def f(a, b)
    return a + b

def mygenerator():
    for x in range(100):
        yield x

mylambda = lambda x: x * 1

myfunc()
a = f(10, 20)
for x in mygenerator():...
mylambda()
```

def – создает новый объект-функцию и присваивает указанной переменной ссылку на этот объект.

lambda – создает новый объект-функцию и возвращает ссылку на него в качестве результата.

return – отправляет результирующий объект вызывающему коду.

yield – отправляет результирующий объект вызывающему коду, но запоминает место, где выполнение остановилось.

Области видимости (scopes)

- **Пространство имен**
  - Место, где существуют (видимы) имена переменные
- **Место создания имени определяет ее пространство имен**
- **Имена, присвоенные внутри def, видимы только в def**
- **Пространства имен**
  - Встроенные имена
  - Модуль (глобальная область видимости)
  - Родительская функция
  - Функция (локальная область видимости)
- **Правило LEGB: local, enclosing, global, built-in**
- **Модификаторы имени (когда надо изм. var в другом scope):**
  - global, nonlocal

Аргументы функции

- **Позиционные и именованные аргументы**
  - `def f(a, b, c, d, e)`
  - Все аргументы обязательные
- **Можно задавать дефалтовые значения (необязательные арг.)**
  - `def f(a, b = 20)`
  - Обязательные аргументы должны быть до необязательных
- **Можно задавать переменное число позиционных аргументов**
  - `def f(a, *b): for arg in b: print(arg)`
- **Можно задавать переменное число именованных аргументов**
  - `def f(**kwargs): for arg in kwargs.keys(): print(arg, kwargs[arg])`
- **В вызове функции можно указывать аргументы по имени:**
  - `def f(a, b = 20): ...` `f(b = 1, a = 2)`
- **Разрешить только фиксированные именнованные аргументы**
  - `def f(a, *, b, c):...` `f(10, b = 20, c = 30)`

Лямбды

- Функция без имени
- lambda [arg1[, arg2, ... argN]]: expression
-

Встроенные функции (функц. прогр.)

- **map**
  - Применяет унарную функцию к каждому элементу последовательности
  - map(func, seq)
  - def f(x): print(x) map(f, [1, 2, 3, 4, 5])
  - map(lambda x: x + 1, [1, 2, 3, 4, 5])
- **reduce (functools)**
  - Применяет бинарную функцию к предыдущему результату и следующему элементу
  - reduce(func, seq, startval)
  - reduce(lambda x, y: x + y, [1,2,3,4,5])
- **filter**
  - Фильтрует входящую последовательность, возвращает новую с отфильтрованными элементами
  - filter(func, seq)
  - filter(lambda x: x % 3 == 0, range(100))
- **apply**
  - Python2: apply(f, ...)
  - Python3: f(*[arglist], **[kwarglist])

Функции - Вопросы

```
def f(): pass
a = f()
print(a)
```

```
def a(a): print(a)
a(10)
```

```
def f(a, *, b): pass
f(10, b = 20, c = 30)
```

```
if x == 2:
    def f():
        print("x == 2")
f()
```

```
a = 10
def f():
     a = a + 1
f()
```

```
def f():
     a = 10
     def g():
          def d():
               print(a)
          d()
     g()
f()
```

```
a = lambda: 1
print(a())
```

```
a = lambda x, y = 2: x + y
print(a())
```

```
(lambda x: print(x))(10)
```

```
f = lambda x, y: [x + y, x * y]
a, b = f(1, 2)
print(a, b)
```

```
a = map(lambda x:x+1, [1,2,3])
print(a)
print(*a)
```

PEP-8 (coding style)

- Использовать 4 пробела для отступа
- Максимальная длина строки 79 символов
- import модулей на разных строках
- Кодировка файла UTF-8
- Не использовать следующие имена (перем из одного символа):
  - ‘l’ (lowercase letter el)
  - ‘O’ (uppercase letter oh)
  - ‘I’ (uppercase letter eye)
- Имена переменных в нижнем регистре
- Имена функций в lowercase+underscore нотации (my_func_1)
- Имена констант заглавными буквами (MAX_OVERFLOW)