ЛЕКЦИЯ 16
Повторение

1. Ссылочная модель данных

Ссылочная модель данных - объекты существуют независимо от времен.

```
>>> 2+3
5
```

Создаются два объекта типа int.

```
2.__add__(3) - метод добавления

x = 2+3 # x ссылается на объект 5
x = 'Hello' # возникает объект строки
```

После вывода результата объекты 2, 3, 5 удаляются, т.к. нет ссылки на эти объекты. Если x перестал ссылаться на 5, объект 5 уничтожается.

```
x = 5
y = x
x = 'Hello'
y = None
```

После того, как y начал ссылаться на None, объект 5 уничтожается.

```
x = (2+3+5)**5**5 # приоритет у возведения в степени
```

T.e. в начале выполняется 5**5, а потом (2+3+5)**25.

Начиная с версии Python 3.6, можно разделять числа так:

```
x = 100_001
y = 0xAF_DC
```

2. Пространство имен

4 пространства: локальные, окружающие, глобальные, строенные (LEGB).

y = 10*x+7 Сначала будет происходить поиск локального x, затем в надпространстве, затем в глобальных, в последнюю очередь - во встроенных переменных.

Если написать max = 10, то функция max перестанет работать.

Пример:

Программа №2.1.

```
1    def f(A):
2        A = A + 10 #если написать A += 10 ошибки не будет
3    B = [1, 2, 3]
4    f(B)
5    print(*B) #1 2 3
```

Строки и числа — неизменяемые объекты. f является именем объекта functional.

def — по сути это операция создания нового объекта.

Любой вызов функции порождает свое пространство имен, которое перестанет существовать после выполнения return.

A начинает ссылаться на [1, 2, 3]. После конкатинации A начинает ссылаться на [1, 2, 3, 4]. A ссылается на глобальный объект, и начинает ссылаться на локальный объект. После return уничтожается A и список [1, 2, 3, 4].

Или можно записать так:

Программа №2.2.

```
1    def f(A):
2        A.append(10)
3    B = [1, 2, 3]
4    f(B)
5    print(*B) #1 2 3
```

Функция должна что-то возвращать. В частном случае, можно возвращать несколько параметров. Нарушить ссылочную модель можно только "залезая" в глобальные имена.

Программа №2.3.

```
1    d=1
2    def f(A):
3        global d
4        A.append(d)
5        d = d + 1
```

3. ООП

Тип тоже является объектом типа тип.

Программа №3.1.

```
1    class Base:
2        x = 10
3        def g(self, x0):
4            self.x = x0
5    b = Base()
6    print(b.x)
```

Программа №3.2.

```
1    class Base:
2        x = 10
3        def g(self, x0)
4            self.x += x0
5    b = Base()
6    print(b.x)
```

Нужно создавать атрибуты только в методе init!

4. Элементы функционального программирования

4.1. Функция map

```
x, y, z = map(int, input().split()) #считывание трех чисел с клавиатуры
```

Функция map применяет к каждому объекту функцию, которую мы написали. В нее же можно написать свою функцию:

```
x, y, z = map(lambda x: int(x)**2, input().split())
A == list(map(int, range(100)))
B = map(float, int(x) for x in input().split())
```

map возвращает объект типа map.

4.2. Применение lambda-функций

$x^2 + e^{1/x} + \ln x$

```
x = decomposed_value # нужно объявлять функцию
(lambda x: x**2 + exp(1/x)+ln(x))(2) #хороший пример использования lambda
```

4.3. Функция enumerate

Программа №4.1.

```
1    A = [10, 20, 30]
2    for i,x in enumerate(A): #A используется как итерируемый объект
3        print(i,x) #(0,10), (1,20), (2,30) - эту конструкцию нам вернет enumerate(A)
4        x = x + 1 #Значение в массиве не изменилось, мы только испортили x
```

4.4. Функция zip

```
A = [1, 2, 3, 4, 5]
B = 'Hello'
c = list(zip(A,B)) #результат есть zip-object
```