



Информатика. Введение в Python

Ссылки. Копирование. Области видимости

*Шевская Наталья Владимировна,
СПбГЭТУ “ЛЭТИ”, ФКТИ, МОЭВМ*



Генераторы списков

```
L = input("Введите список чисел через пробел").split(' ')
M_for = []
for i in range(10):
    M_for.append(int(i))

M_gen = [int(x) for x in L]

print(M_for)
print(M_gen)
```



Генераторы списков с условием



```
M = [x for x in range(10) if not x % 2]  
print(M)
```

```
L = input("Введите строки через пробел: ").split(' ')  
M = [item for item in L if item[0].isupper()]  
print(M)
```



Матрицы на основе списка

```
matrix = [[0, 0, 0],  
           [0, 0, 0],  
           [0, 0, 0]]
```

➤ Создание матрицы с помощью генератора списка:

```
matrix = [[0 for x in range(3)] for y in range(3)]
```

элемент списка -- список



Генераторы словарей

```
res_for = {}
```

```
for x in 'QwErTyQ':
```

```
    res_for.update({x: ord(x)})
```

```
res_gen = {x: ord(x) for x in 'QwErTyQ'}
```

```
res_gen = {key: value
```

```
    for key, value in enumerate('QwErTyQ')}
```

```
print(res_gen)
```



Генераторы словарей с условием



```
res_gen = {key: ord(key) for key in 'QwErTyQ'  
            if key.isupper()}  
print(res_gen)
```

```
res = {key: value for key, value in enumerate('QwErTyQ')  
        if value == "Q" and not key % 3}  
print(res)
```

Связывание имени с объектом



Операция присваивания

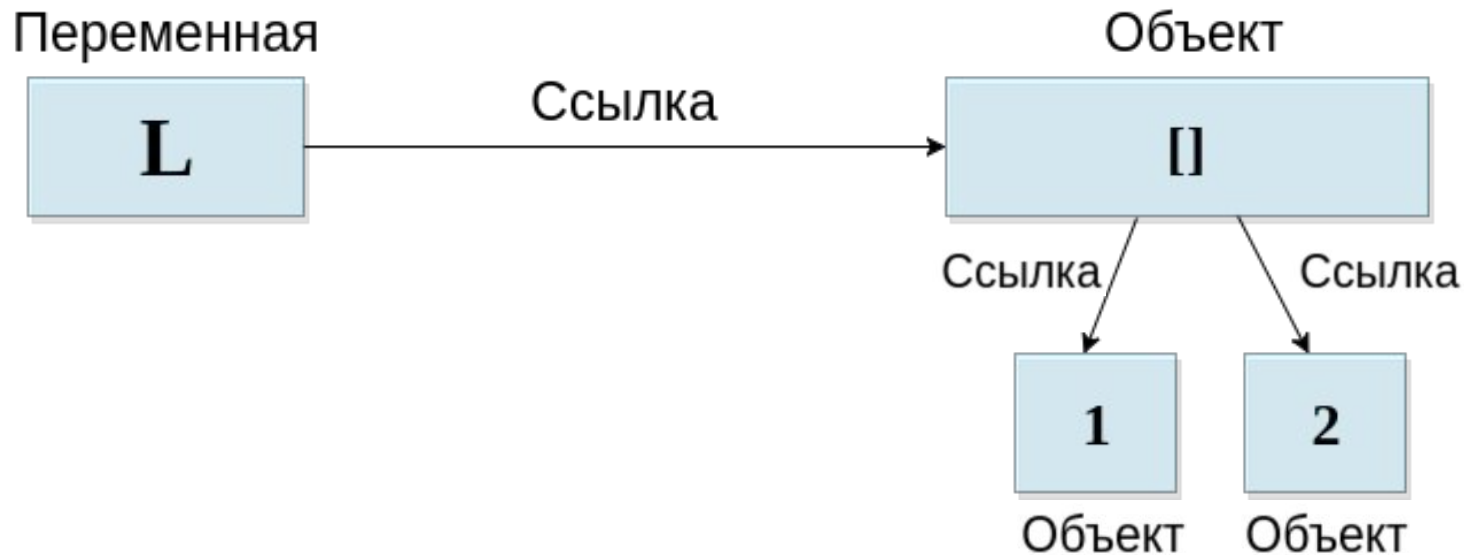
x = 100





Операция присваивания

`L = [1, 2]`





Операция присваивания



```
x = 100
```

Действия интерпретатора:

- Создается объект 100
- Переменная x создается, если до этой строки она не была создана
- В переменную x записывается ссылка на объект 100



Операция присваивания. Кэширование

```
x = 100
```

```
y = 100
```

```
print(id(x))
```

```
print(id(y))
```

- x и y указывают на одну и ту же область памяти,
- объект 100 -- единственный в памяти (кэширование)



Переменные, объекты, ссылки



x = 100

- **Переменная** хранит ссылку на объект
- **Объект** - область памяти, которая хранит значение объекта. Имеет два стандартных поля: тип и счетчик ссылок на этот объект
- **Ссылка** - это автоматически разыменовываемый указатель на объект



Разделяемые ссылки неизменяемых объектов

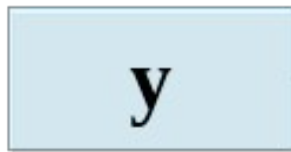
```
x = 100
```

```
y = x
```

Переменная



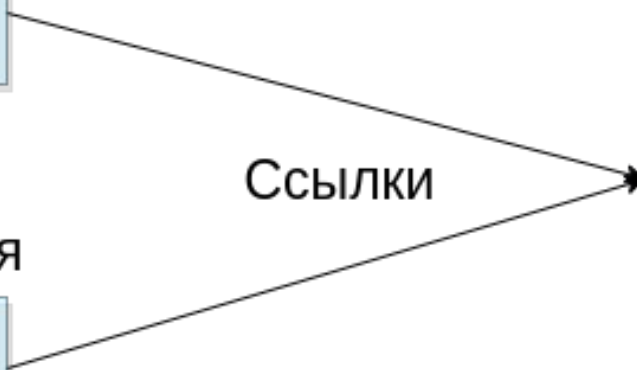
Переменная



Объект



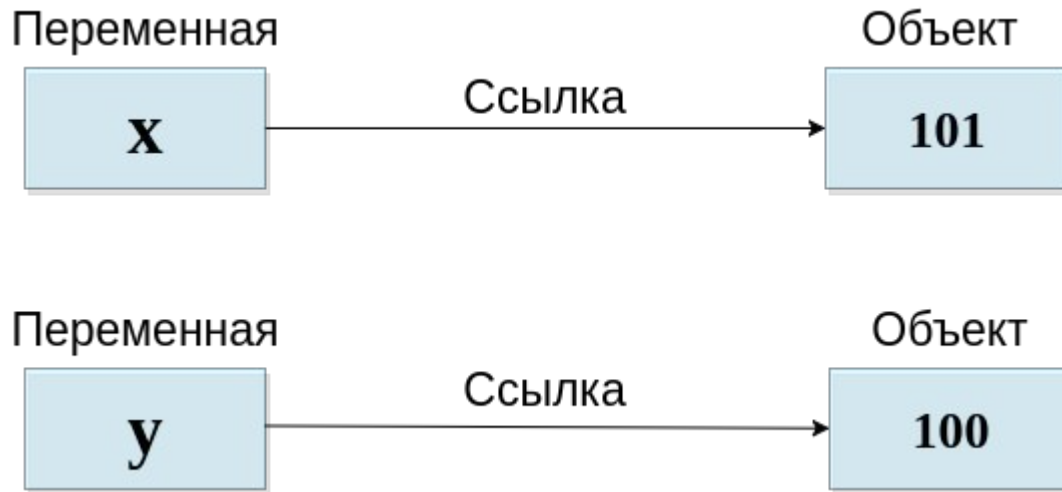
Ссылки





Разделяемые ссылки неизменяемых объектов. Копирование

```
x = 100  
y = x  
x += 1
```

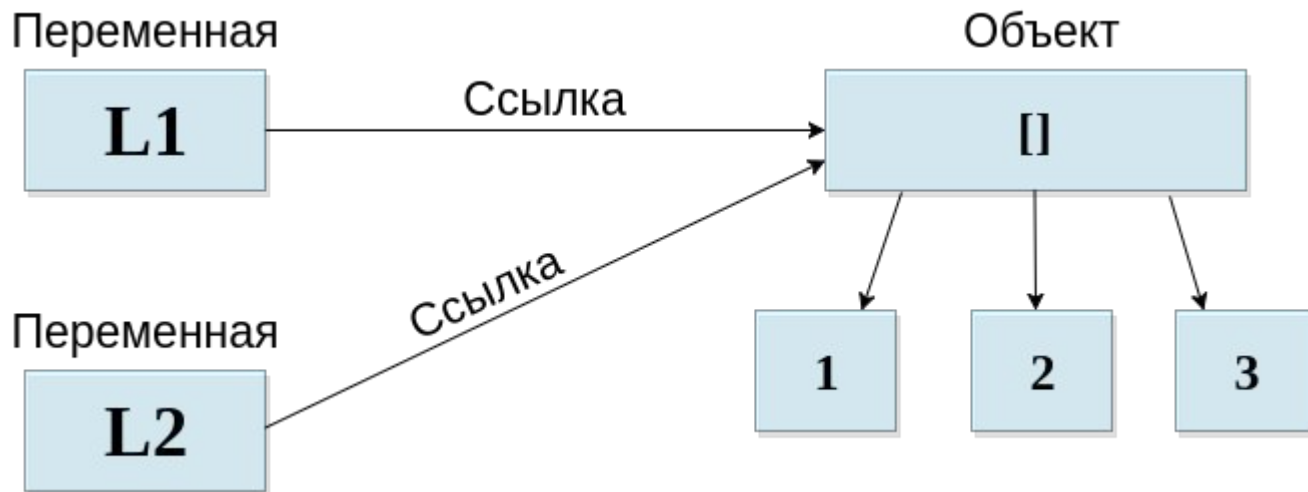




Разделяемые ссылки изменяемых объектов

```
L1 = [1, 2, 3]
```

```
L2 = L1
```



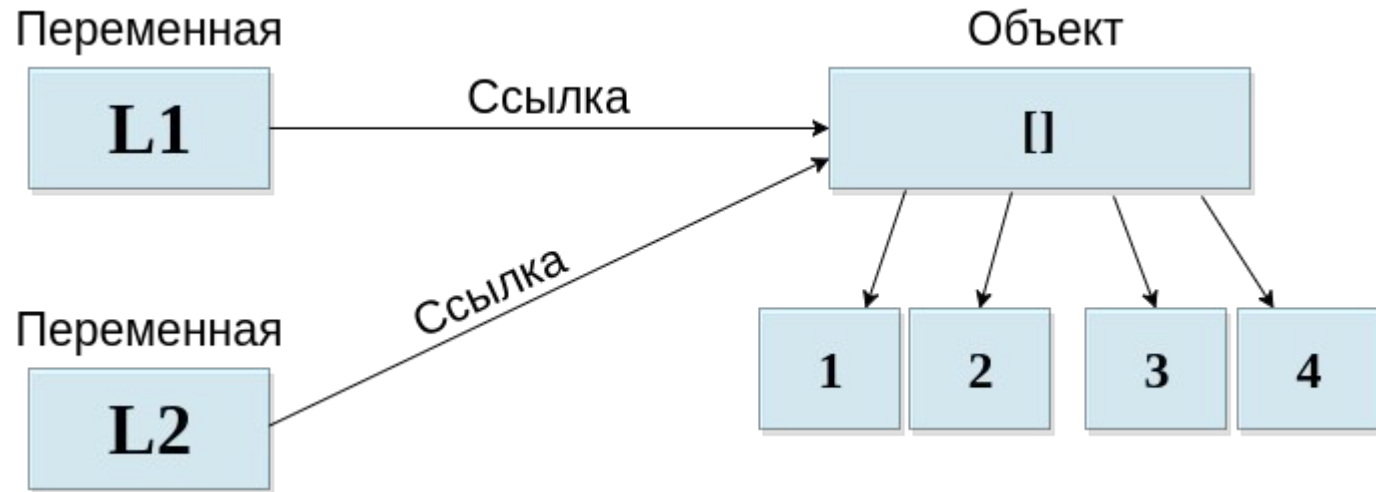


Разделяемые ссылки изменяемых объектов (2)

```
L1 = [1, 2, 3]
```

```
L2 = L1
```

```
L1.append(4)
```





Проверка идентичности объектов.

Операция **is**

```
a = 100900
```

```
b = 100900
```

```
a == b # проверка на равенство значений объектов
```

```
a is b # проверка на идентичность объектов
```

```
a is not b # проверка на то, что объекты разные
```



Изменяемые и неизменяемые объекты

Неизменяемые:

- числа
- строки
- логический тип
- кортежи

Изменяемые:

- списки
- словари
- множества



Копирование списка

создание поверхностной копии списка b

```
a = list(b)
```

создание поверхностной копии списка b

```
a = b[:]
```

создание поверхностной копии списка b

```
a = b.copy()
```



Модуль для копирования списка



Модуль `copy`

`import copy`

- метод `copy()`:

`a = copy.copy(b)` # создание поверхностной копии списка `b`

- метод `deepcopy()`:

`a = copy.deepcopy(b)` # создание полной копии списка `b`



Передача аргументов в функцию

- Аргументы передаются через автоматическое **присваивание** объектов локальным переменным.
 - Изменение **изменяемого** объекта внутри функции приведет к изменению объектов в вызывающей программе.
 - Изменение **неизменяемого** объекта внутри функции не приведет к изменению объектов в вызывающей программе.



Области видимости переменных



- **Область видимости** - это место в коде, где определяются имена и где они могут быть найдены.
- **Пространство имён** - место, где находятся имена.
- Любая функция содержит в себе **пространство имён**: имена, определяемые внутри функции, видны только в пределах этой функции.
- Функции образуют **локальную** область видимости, а модули – **глобальную**.



Области видимости переменных (2)



- Если присваивание происходит внутри функции, переменная является **локальной** для этой функции.

```
x = 100
def f():
    x = 777
    return(x)
print(f())
print(x)
```



Области видимости переменных (3)



- Если присваивание происходит в пределах некоторой внешней функции, переменная является **нелокальной** для внутренней функции.

```
def f2(): # внешняя функция
    def f1(): # внутренняя функция
        nonlocal x # без nonlocal нельзя изменить x
        x = 100
        return x
    x = 777
    print(f1())
    return x
print(f2())
```




Области видимости переменных



- Если присваивание производится за пределами всех инструкций **def**, переменная является **глобальной** для всего файла.

```
def f1():  
    global x  
    x = 100
```

```
x = 777  
print(x)  
f1()  
print(x)
```



Области видимости переменных. LEGB

области видимости

B -- built-in -- встроенные/зарезервированные имена

G -- global -- на самом верхнем уровне

E -- enclosed -- концепция вложенных функций

L -- local -- внутри функции или класса*



Разыменование (повторение)



```
a = 100900
```

```
b = a
```

ссылка с именем **a**, которая стоит справа от присваивания,
автоматически разыменовывается -- теперь имя **b** указывает на ту же область памяти, что и **a**



Присваивание

a = **100900**

b = **a**

присваивание выполняет роль *связывания* имени **a**
со ссылкой на конкретную область памяти, куда
записывается объект **100900**



Как работает память

Visualize Python, Java, JavaScript, C, C++, Ruby code execution

Неизменяемые типы:

Python 3.6
(known limitations)

```
1 x = 2
2 y = x
→ 3 x += 1
```

[Edit this code](#)

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

Done running (3 steps)

Frames

Global frame	
x	3
y	2

Objects

Изменяемые типы:

Python 3.6
(known limitations)

```
1 L = [1, 2, [3, 2, 1]]
2 M = L
→ 3 M[2][0] += 1
```

[Edit this code](#)

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

Done running (3 steps)

Frames

Global frame	
L	→
M	→

Objects

list			
0	1	2	
1	2	→	

list			
0	1	2	
4	2	1	



Полезные ссылки по 1-й части

1. Курс на Stepik “Python: основы и применение” уроки 1.2, 1.4, 2.2
<https://stepik.org/course/512/syllabus>
2. Learning Python, Fourth Edition, by Mark Lutz. Copyright 2009 O'Reilly Media, Inc., 978-0-596-15806-4
3. Вики от института биоинформатики, области видимости:
http://wiki.bioinformaticsinstitute.ru/wiki/%D0%9E%D0%B1%D0%BB%D0%B0%D1%81%D1%82%D0%B8_%D0%B2%D0%B8%D0%B4%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D0%B8
4. Документация Python, генераторы списков
<https://docs.python.org/3/howto/functional.html#generator-expressions-and-list-comprehensions>

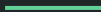


PyCharm

чуть больше про IDE



- автодополнение
- проверка до исполнения программы
- форматирование по PEP8
- горячие клавиши
- ...





Умный редактор кода

Все горячие клавиши приведены для Linux.

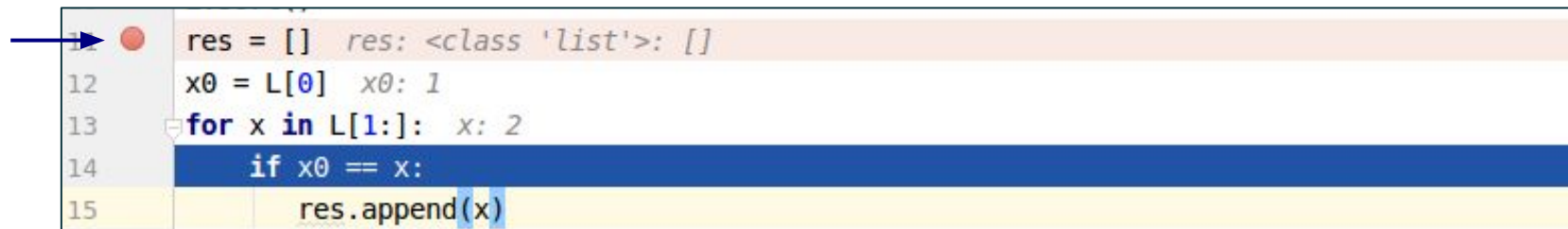
- Автодополнение кода (контекстно-чувствительное)
- Документация по элементу, на котором находится курсор **Ctrl + Q**
- Показать описание ошибки **Ctrl + F1 [+ Fn]**
- Форматирование кода согласно PEP8 **Ctrl + Alt + L**
- Перейти к определению метода **Ctrl + щелчок мыши на вызове метода**
- Посмотреть все использования метода **Ctrl + щелчок мыши на определении метода**
- Изменение прототипа функции во всем коде **Ctrl + F6 [+ Fn]**
- Изменение названия объекта **Shift + F6 [+ Fn]**

help -> Keymap Reference # получить pdf со всеми основными сочетаниями клавиш



Отладчик

Breakpoints (точки останова) позволяют приостановить выполнение программы в определенной точке и изучить ее поведение.



```
11  res = []  res: <class 'list'>: []  
12  x0 = L[0]  x0: 1  
13  for x in L[1:]:  x: 2  
14      if x0 == x:  
15          res.append(x)
```

The screenshot shows a code editor with a Python script. A red circle breakpoint is set on line 11. A blue arrow points to the breakpoint. The code is as follows:



Отладчик

Step Into

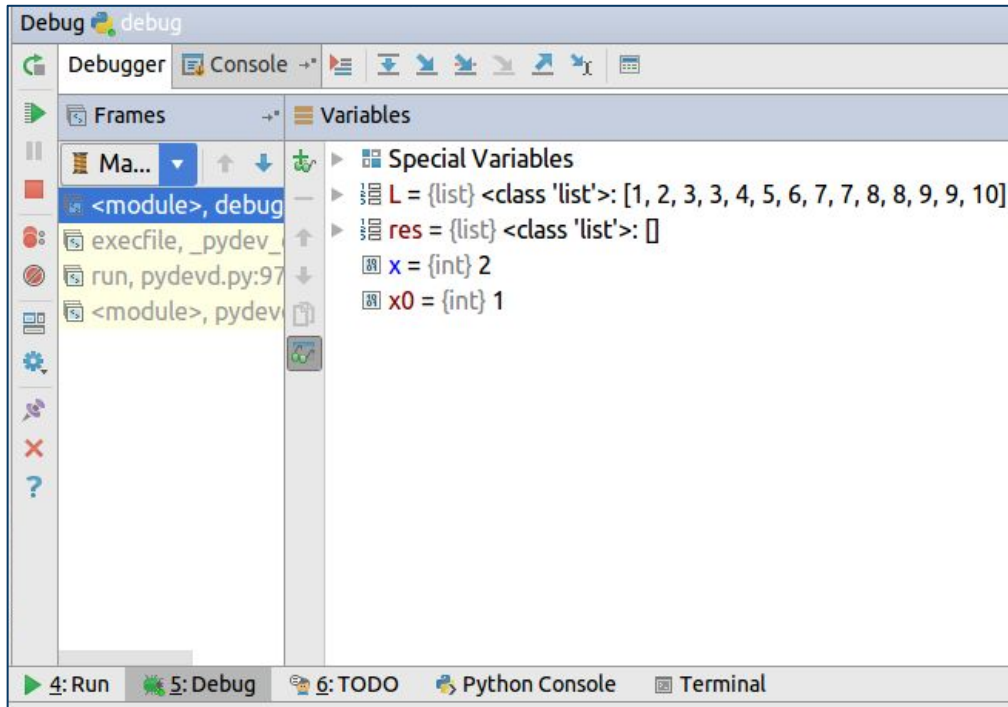
Позволяет зайти в метод в момент вызова

Step Into My Code

Позволяет не заходить в библиотечные методы, только в методы пользователя

Step Over

Позволяет “перепрыгивать” через вызов метода, игнорируя точки останова на пути.





Полезные ссылки про PyCharm

- Особенности и возможности PyCharm:
<https://www.youtube.com/watch?v=DpscmxH2LQU>
- Stepping through the program
<https://www.jetbrains.com/help/pycharm/stepping-through-the-program.html>
- Breakpoints <https://www.jetbrains.com/help/pycharm/using-breakpoints.html>
- Пример
<https://www.jetbrains.com/help/pycharm/debugging-your-first-python-application.html>
- Работа с git в Pycharm:
https://github.com/moevm/cs_lectures/blob/master/KC4_git_and_collaboration.pdf

Вопросы по курсу задавайте по почте:

Шевская Наталья Владимировна
natalya.shevskaya@moevm.info

Правила коммуникации по электронной почте:

http://se.moevm.info/doku.php/inf:communication_rules