

ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ

КАФЕДРА АВТОМАТИЗАЦИЯ ПРОИЗВОДСТВЕННЫХ ПРОЦЕССОВ

ЛЕКЦИЯ №2

Инструкции и синтаксис языка  
программирования Python и введение  
в стандарт PEP 8

СОСТАВИТЕЛЬ:            КАНД. ТЕХН. НАУК    БЫКАДОР В.С.

# Отступы являются синтаксисом языка Python

Отступы в языке программирования Python указывают на блок кода.

*C*

```
...  
...  
for (uint8_t i = 0; i < 16; i++)  
{  
    PORTB &= 0xFE;  
    PORTB |= (data >> i) << SDI;  
    switch_pin_(SFTCLK);  
}
```

*Pascal*

```
...  
...  
for i:=0 to N do  
begin  
    a := a + 1;  
    b := a/2;  
end;
```

*Python*

```
...  
...  
for a in mylst:  
    a = a + 1;  
    newlst.append(a);
```



Отступы в языке программирования Python ОБЯЗАТЕЛЬНЫ!

# Основное о PEP8

**PEP8** является обще принятым стандартом оформления кода, написанного на языке программирования Python. Существует большое количество рекомендаций о том, как нужно оформлять код, написанный на Python.

Рассмотрим лишь основные из них:

- 1) Имена переменных и функций пишут строчными буквами.
- 2) Если имя состоит из более чем одного слова, то их разделяют подчёркиванием.
- 3) Имена классов пишут в стиле камелкейс (CamelCase).
- 4) Между импортом модулей и первой строкой кода оставляют две пустые строки.
- 5) Между функциями оставляют по одной пустой строке.
- 6) После последней строки кода в файле оставляют пустую строку.

# Основное о PEP8

```
3 from clsDiagnosis import Diagnosis
4 from clsDiagnosticObject import DiagnosticObject
5 from clsMetricAnalysis import MetricAnalysis
```

```
6
7
8 number_variant = '06'
9
10 lambdas = {
11     'lambda 1': 0.14
12     , 'lambda 2': 0.3
13     , 'lambda 3': 0.4
14     , 'lambda 4': 0.16
15 }
```

```
102 class LoadData:
103     def __init__(self, num_variant, xml_file_variants, xml_file_data):
104         """
105         :param num_variant: номер варианта задания (две последние цифры заче
106         :param xml_file_variants: путь к xml-файлу со схемой задания
107         :param xml_file_data: путь к xml-файлу со числовыми данными
108         """
109         self.__num_var = num_variant
```

```
91 def get_data_to_X(self):
92     """
93     Возвращает вектора объектов распознавания Xi
94     :return: список списков координат объектов распознавания
95     """
96
97     return self.get_data_to_D('X')
```

```
98
```

# Условный оператор `if`

Типовая конструкция

```
>>> i = True
>>>
>>> if i == True:
...     print(':))')
... else:
...     print(':(')
...
... :))
>>>
```

Конструкция  
множественного выбора

```
>>>
>>> i = 2
>>>
>>> if i == 1:
...     print('a')
... elif i == 2:
...     print('b')
... elif i == 3:
...     print('c')
... else:
...     print('иначе')
...
b
>>>
```

**Обратите внимание на двоеточия и обязательные отступы блоков кода!**

# Тернарная форма условной операции `if`

Существует тернарная форма условной операции, которая позволяет условную операцию, у которой в блоках кода по одной команде, записывать в более компактной форме:

**<истина> if <условие> else <лож>**

Типовая конструкция

```
>>> i = True
>>>
>>> if i == True:
...     print(':))')
... else:
...     print(':(')
...
... :))
>>>
```

Тернарная форма условной операции

```
>>>
>>> print(':))') if i == True else print(':(')
>>> :))
>>>
```

# Тернарная форма условной операции `if`

```
>>>  
>>> result = ':))' if i else ':( '  
>>> print(result)  
:))  
>>>
```

В переменную `result` записывается результат операции `if` и затем полученный результат выводится при помощи функции `print()`. При этом обратите внимание на то, что нет необходимости явно прописывать условие `if i == True`, так как по умолчанию выполняется проверка на «истинность» условия. То есть записи: `if i == True` и `if i` являются эквивалентными.

# Операции сравнения и логические операции

## Операции сравнения

<code>==</code>	равно
<code>!=</code>	неравно
<code>&gt;</code>	больше
<code>&lt;</code>	меньше
<code>&gt;=</code>	больше или равно
<code>&lt;=</code>	меньше или равно

## Логические операции

<code>and</code>	«И» - конъюнкция
<code>or</code>	«ИЛИ» - дизъюнкция
<code>not</code>	«НЕ» - отрицание

```
>>> str1 = 'a'
>>> str2 = 'b'
>>>
>>> if str1 == 'a' and str2 == 'a':
...     print(True)
... else:
...     print(False)
...
False
>>>
>>>
>>> if str1 == 'a' and not(str2 == 'a'):
...     print(True)
... else:
...     print(False)
...
True
>>>
```

```
>>> print(1<=0)
False
>>> print(1<=1)
True
>>>
```



# Оператор цикла `for`

Оператор цикла `for` выполняет перебор элементов коллекции, начиная с первого элемента и до последнего элемента.

```
>>>
>>> lst = [1, 'привет', 2.55]
...
>>> for item in lst:
...     print(item)
...
...
1
привет
2.55
>>>
```

Обратите внимание на двоеточия и обязательные отступы блоков кода!

# Некоторые приёмы работы с оператором цикла `for`

Оператор цикла `for` выполняет перебор элементов коллекции, начиная с первого элемента и до последнего элемента.

```
0
7  lst_last_name = ['Иванов', 'Петров', 'Сидорова']
8  lst_first_name = ['Сергей', 'Станислав', 'Ирина']
9
10 for i in range(len(lst_first_name)):
11     print(f'{lst_first_name[i]} {lst_last_name[i]}')
12
13
14
```

ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ    JUPYTER

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python _lectures.py
```

```
Сергей Иванов
Станислав Петров
Ирина Сидорова
```

```
6
7  lst_last_name = ['Иванов', 'Петров', 'Сидорова']
8  lst_first_name = ['Сергей', 'Станислав', 'Ирина']
9
10 for first_name, last_name in zip(lst_first_name, lst_last_name):
11     print(f'{first_name} {last_name}')
12
13
```

ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ    JUPYTER

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python _lectures.py
```

```
Сергей Иванов
Станислав Петров
Ирина Сидорова
```

Функция `range(n)` формирует диапазон 0 ... n-1

```
print(list(range(5)))
```

```
[0, 1, 2, 3, 4]
```

```
• (venv) vityalybykador@Air-Vitaly test %
```

Функция `zip(n)` формирует общий итератор из нескольких итераторов (списки, кортежи).

# Генератор как сокращенная форма оператора `for`

Генератор позволяет получить сокращенную форму оператора цикла `for` для заполнения списка `list()`.

`<list> = [element for element in some_list]`

```
7 numbers = [1, 3, 5, 10]
8 result = []
9
10 for num in numbers:
11     result.append(num**2)
12
13 print(result)
14
15
```

ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ

• (venv) vityalybykador@Air-Vitaly test % /Users/vityalybykador/\_lectures.py

[1, 9, 25, 100]

```
7 numbers = [1, 3, 5, 10]
8
9 result = [num**2 for num in numbers]
10
11 print(result)
12
13
14
15
```

ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ

• (venv) vityalybykador@Air-Vitaly test % /Users/vityalybykador/\_lectures.py

[1, 9, 25, 100]

# Генератор с условием

```
7 numbers, result = [1, 3, 5, 10], []
8 for num in numbers:
9     if num < 5:
10         result.append(num**2)
11
12 print(result)
13
14
15
```

ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ

• (venv) vitalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/\_lectures.py

[1, 9]

```
7 numbers = [1, 3, 5, 10]
8
9 result = [num**2 for num in numbers if num < 5]
10
11 print(result)
12
13
```

ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ    JUPYTER

• (venv) vitalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/\_lectures.py

[1, 9]

# Оператор цикла `while`

Если требуется выполнять циклические действия до тех пор, пока не наступит какое-либо условие, то для этих целей подойдёт оператор цикла **`while`**.

Например, будет выполняться возведение чисел в **квадрат**, но до тех пор, пока мы не получим число равное или больше **25**. Как мы получим число **равное или больше 25**, то прекращаем выполнение цикла.

```
5
6
7 numbers, i, local, result = [1, 3, 4.4, 5, 10], 0, 0, list()
8
9 while local < 25:
10     local = numbers[i]**2
11     result.append(local)
12     i = i + 1
13
14 print(result)
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
• (venv) vitalitybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/py
PROJECTS/test/for_lectures.py
```

[1, 9, 19.360000000000003, 25]

`i += 1`

# Множественный выбор

Существует несколько вариантов реализации множественного выбора в языке программирования Python.

Вариант № 01 - через оператор `if`:

```
6  eng = 'What'
7  rus = ''
8
9  if eng == 'Hello':
10 |     rus = 'Привет'
11 elif eng == 'What':
12 |     rus = 'Что'
13 elif eng == 'goodbye':
14 |     rus = 'Пока'
15 else:
16 |     rus = 'Ошибка'
17
18 print(rus)
19
```

PROBLEMS OUTPUT DEBUG CONSOLE

```
• (venv) vityalybykador@Air-Vitaly test 9
PROJECTS/test/for_lectures.py
```

Что

можно короче

```
5
6  eng = 'goodbye'
7  rus = 'Ошибка'
8
9  if eng == 'Hello':
10 |     rus = 'Привет'
11 if eng == 'What':
12 |     rus = 'Что'
13 if eng == 'goodbye':
14 |     rus = 'Пока'
15
16
17
18 print(rus)
19
```

PROBLEMS OUTPUT DEBUG CONSOLE

```
• (venv) vityalybykador@Air-Vitaly test
PROJECTS/test/for_lectures.py
```

Пока

# Множественный выбор

Существует несколько вариантов реализации множественного выбора в языке программирования Python.

Вариант № 02 - оператор `match`:

Только с Python 3.10

`match subject:`

`case pattern1:`

`action1`

`case pattern2:`

`action_2`

`case pattern_3:`

`action_3`

`case _:`

`default_action`

```
10 eng = 'goodbye'
11 rus = ''
12
13 match eng:
14     case 'Hello':
15         rus = 'Привет'
16     case 'What':
17         rus = 'Что'
18     case 'goodbye':
19         rus = 'Пока'
20     case _:
21         rus = 'Ошибка'
22
23 print(rus)
```

PROBLEMS OUTPUT DEBUG CONSOLE

(venv) vityalybykador@Air-Vitaly test %  
PROJECTS/test/for\_lectures.py

Пока

```
10 eng = '111111'
11 rus = 'Ошибка'
12
13 match eng:
14     case 'Hello':
15         rus = 'Привет'
16     case 'What':
17         rus = 'Что'
18     case 'goodbye':
19         rus = 'Пока'
20
21
22
23 print(rus)
```

PROBLEMS OUTPUT DEBUG CONSOLE

(venv) vityalybykador@Air-Vitaly test  
PROJECTS/test/for\_lectures.py

Ошибка

# Множественный выбор

```
11 value = 1
12 res = None
13
14 match value:
15     case int() if value < 10:
16         res = value + 10
17     case int() if not(value < 10):
18         res = value - 10
19     case float():
20         res = value**2
21     case str():
22         res = value.upper()
23     case list():
24         res = [v for v in value if (v is not None) and isalpha(v)]
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Use  
PROJECTS/test/for\_lectures.py

11

```
10
11 value = 100
12 res = None
13
14 match value:
15     case int() if value < 10:
16         res = value + 10
17     case int() if not(value < 10):
18         res = value - 10
19     case float():
20         res = value**2
21     case str():
22         res = value.upper()
23     case list():
24         res = [v for v in value if (v is not None) and isalpha(v)]
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Use  
PROJECTS/test/for\_lectures.py

90



# Множественный выбор

```
10
11 value = 2.4
12 res = None
13
14 match value:
15     case int() if value < 10:
16         res = value + 10
17     case int() if not(value < 10):
18         res = value - 10
19     case float():
20         res = value**2
21     case str():
22         res = value.upper()
23     case list():
24         res = [v for v in value if (v is not None) and isalpha(v)]
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Use  
PROJECTS/test/for\_lectures.py

5.76

```
11 value = 'привет!'
12 res = None
13
14 match value:
15     case int() if value < 10:
16         res = value + 10
17     case int() if not(value < 10):
18         res = value - 10
19     case float():
20         res = value**2
21     case str():
22         res = value.upper()
23     case list():
24         res = [v for v in value if (v is not None) and isalpha(v)]
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

(venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Use  
PROJECTS/test/for\_lectures.py

ПРИВЕТ!

# Множественный выбор

```
11 value = [1, 3.4, 'a', None, 'b'] ←
12 res = None
13
14 match value:
15     case int() if value < 10:
16         res = value + 10
17     case int() if not(value < 10):
18         res = value - 10
19     case float():
20         res = value**2
21     case str():
22         res = value.upper()
23     case list():
24         res = [v for v in value if (v is not None) and isalpha(v)]
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Users/PROJECTS/test/for\_lectures.py

['a', 'b']

# Множественный выбор

Существует несколько вариантов реализации множественного выбора в языке программирования Python.

Вариант № 03 - через словарь `dict()`:

```
6 dict_eng_rus = {
7     'Hello': 'Привет',
8     'What': 'Что',
9     'goodbye': 'Пока'
10 }
11
12 eng = 'Hello'
13 rus = 'Ошибка'
14
15 if eng in dict_eng_rus.keys():
16     rus = dict_eng_rus[eng]
17
18 print(rus)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
• (venv) vitalitybykador@Air-Vitaly test % /Users/vi
PROJECTS/test/for_lectures.py
```

Привет

```
6 dict_eng_rus = {
7     'Hello': 'Привет',
8     'What': 'Что',
9     'goodbye': 'Пока'
10 }
11
12 eng = 'somethin'
13 rus = 'Ошибка'
14
15 if eng in dict_eng_rus.keys():
16     rus = dict_eng_rus[eng]
17
18 print(rus)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
(venv) vitalitybykador@Air-Vitaly test % /Users/vi
PROJECTS/test/for_lectures.py
```

Ошибка

# Множественный выбор

```
11 a, b, c = 2, 4, None
12 operation = '*'
13
14 match operation:
15     case '+': c = a + b
16     case '-': c = a - b
17     case '*': c = a * b
18
19 print(f'c = {c}')
20
```

PROBLEMS OUTPUT DEBUG CONSOLE

```
(venv) vityalybykador@Air-Vitaly test %  
PROJECTS/test/for_lectures.py
```

c = 8

```
5 a, b, c = 2, 4, None
6 operation = '-'
7
8 dict_op = {
9     '+': lambda x, y: x + y,
10    '-': lambda x, y: x - y,
11    '*': lambda x, y: x * y
12 }
13
14 func_ = dict_op[operation]
15 c = func_(a, b)
16 print(f'c = {c}')
17
18 print('')
19 print('')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
(venv) vityalybykador@Air-Vitaly test % /Users/v  
PROJECTS/test/for_lectures.py
```

c = -2