

Лекция 5. Исключения. Dataclass

Ошибки при написании кода

Ошибка — это неправильное или неточное действие, в результате которого ваша программа либо не запускается вовсе, либо выдает неправильный ответ.

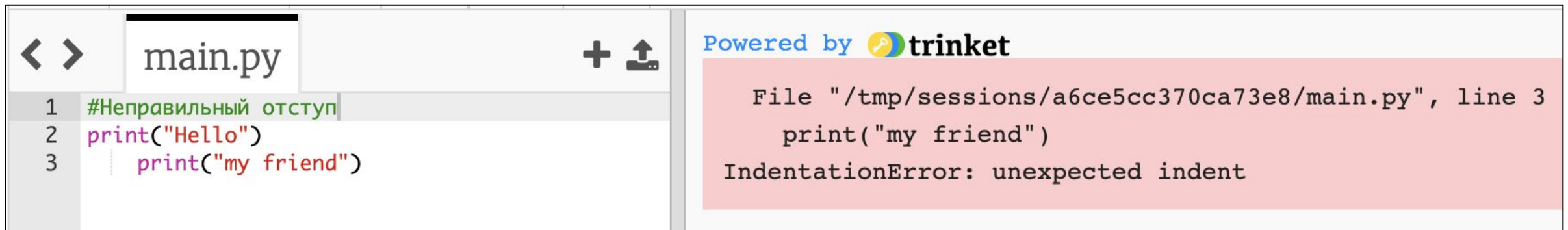
Типы ошибок:

- Синтаксические ошибки;
- Исключения;
- Логическая ошибка.

Синтаксическая ошибка

Синтаксическая ошибка возникает, когда мы не следуем правильной структуре или синтаксису языка.

При запуске программы, содержащей синтаксическую ошибку, Python сообщит вам, что ваш код не работает, и выдаст информацию об ошибке.

The image shows a screenshot of a web-based Python IDE. On the left, a code editor window titled 'main.py' contains three lines of Python code. Line 1 is a comment: '#Неправильный отступ'. Line 2 is 'print("Hello")'. Line 3 is ' print("my friend")', where the indentation is visually incorrect. On the right, a red error message box displays the message: 'File "/tmp/sessions/a6ce5cc370ca73e8/main.py", line 3' followed by 'print("my friend")' and 'IndentationError: unexpected indent'. The IDE interface includes navigation icons and a 'Powered by trinket' logo.

При этом программа, в которой есть синтаксические ошибки, даже не запустится. Python о них сообщит сразу.

Логические ошибки →

Логическая ошибка

Программа запускается и в процессе выполнения не выдает ошибку и может что-то выводить, но ответ не совпадает с тем, который должен быть.

< >

main.py

+

↑

1

#неправильно подсчитывается количество долларов, которое можно приобрести

2


dollar_exchange_rate_25_05_22 = 60.35

3

rubles = 1800

4

print(f'Вы можете купить {rubles * dollar_exchange_rate_25_05_22} долларов')

Powered by  trinket

Вы можете купить 108630.0 долларов

Исключения

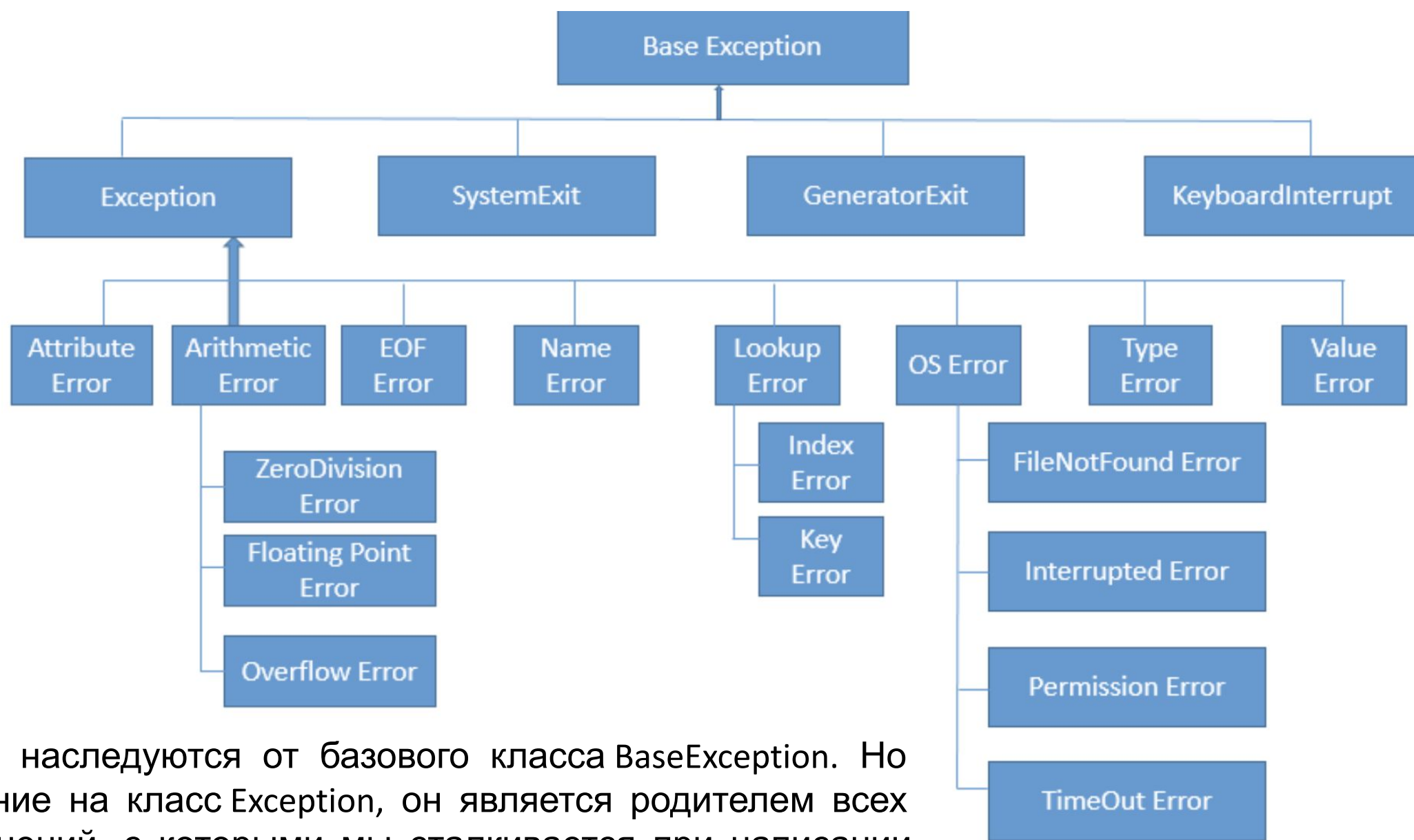
Исключение – это событие, возникающее во время исполнения программы, нарушающее нормальный ход выполнения (например, исключение `KeyError` возникает, когда ключ не найден в словаре). Исключение — это объект Python, представляющий ошибку.

В Python исключение — это объект, производный от класса `BaseException`, содержащий информацию об ошибке, произошедшей в вашей программе.

Объект исключения включает в себя:

- Тип ошибки (имя исключения)
- Состояние программы в момент возникновения ошибки
- Сообщение об ошибке описывает событие ошибки.

Наследование исключений →



Все исключения наследуются от базового класса BaseException. Но обратите внимание на класс Exception, он является родителем всех основных исключений, с которыми мы сталкиваемся при написании программ.

Обработка исключений
→

Обработка исключений

По названию каждого типа исключения можно догадаться, в какой момент данное исключение может произойти. К примеру, исключение `NameError` возникает при обращении к несуществующей переменной.

< >

main.py

+

⬆

1

2

3


4

`print('1')`

`print('2')`

`print(value)`

`print('3')`

Powered by  trinket

1

2

Traceback (most recent call last):


File `"/tmp/sessions/555e168b9dd5ba0b/main.py"`, line 3, in `<module>`

`print(value)`

`NameError: name 'value' is not defined. Did you mean: 'False'?`


Пример →

```
< > main.py + ↗
1 #На ноль делить нельзя
2 print(1)
3 print(2)
4 print(1/0)
5 print(3)
```

Powered by  trinket

```
1
2
Traceback (most recent call last):
  File "/tmp/sessions/655800c397d51788/main.py", line 3, in <module>
    print(1/0)
ZeroDivisionError: division by zero
```

```
< > main.py + ↗
1 print(1)
2 print(2)
3 try:
4     print(3)
5     print(1/0)
6     print(4)
7 except ZeroDivisionError:
8     print('Ошибка деления на ноль!')
9 print(5)
```

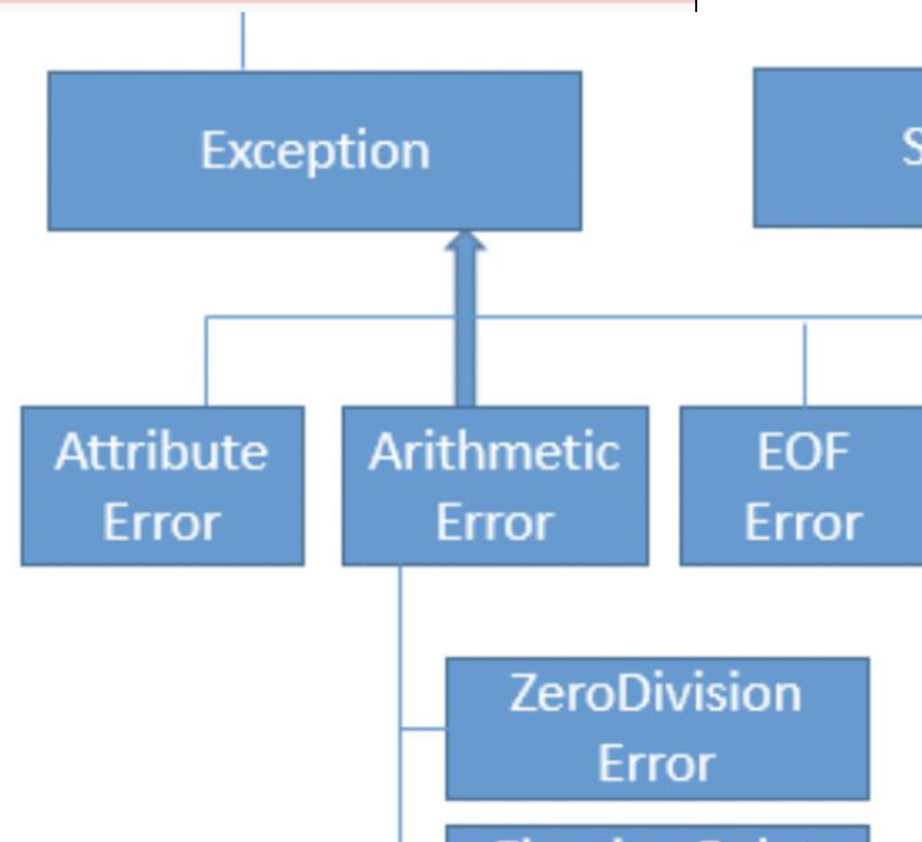
Powered by  trinket

```
1
2
3 Ошибка деления на ноль!
5
```

```
> main.py + ↗
1 print(1)
2 print(2)
3 try:
4     print(3)
5     print(1/0)
6 except ArithmeticError:
7     print('Ошибка ArithmeticError!')
8 print(4)
```

Powered by  trinket

```
1
2
3 Ошибка ArithmeticError!
4
```



Пример →

< >

main.py

+

```
1 print(1)
2 print(2)
3 try:
4     print(3)
5     print(1/0)
6 except ArithmeticError as ex:
7     print(f'Ошибка {ex}!', type(ex))
8 print(4)
```

Powered by trinket

1
2
3
Ошибка division by zero! <class 'ZeroDivisionError'>
4

< >

main.py

+

```
1 try:
2     number = int(input())
3 except:
4     raise ValueError('Неправильный тип данных')
```

Powered by trinket

45

< >

main.py

+

```
1 try:
2     number = int(input())
3 except:
4     raise ValueError('Неправильный тип данных')
```

Powered by trinket

4.2

Traceback (most recent call last):
 File "/tmp/sessions/49dc0ec8a366f1f6/main.py", line 2, in <module>
 number = int(input())
ValueError: invalid literal for int() with base 10: '4.2'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
 File "/tmp/sessions/49dc0ec8a366f1f6/main.py", line 4, in <module>
 raise ValueError('Неправильный тип данных')
ValueError: Неправильный тип данных

Распространение исключений

→

Распространение исключений

main.py

```
1 def second_func():
2     print('Начало работы функции second_func 1')
3     1/0
4     print('Конец работы функции second_func 2')
5
6
7 def first_func():
8     print('Начало работы функции first_func 3')
9     second_func()
10    print('Конец работы функции first_func 4')
11
12
13 print(1)
14 print(2)
15 first_func()
16 print(3)
```

Powered by trinket

```
1
2
Начало работы функции first_func 3
Начало работы функции second_func 1


Traceback (most recent call last):
  File "/tmp/sessions/8c3ea94dbf051c14/main.py", line
15, in <module>
    first_func()
  File "/tmp/sessions/8c3ea94dbf051c14/main.py", line
9, in first_func
    second_func()
  File "/tmp/sessions/8c3ea94dbf051c14/main.py", line
3, in second_func
    1/0
ZeroDivisionError: division by zero
```

Строка вызвавшая **исключение** распространяет **исключение** в
↓
функцию (модуль), которая его вызвала и дальше в
↓
основную программу (main)

Решение→

Конструкция try-except

```
< > main.py +   
1 def second_func():  
2     print('Начало работы функции second_func 1')  
3     try:  
4         1 / 0  
5     except:  
6         print('Внимание! Обработано исключение!')  
7     print('Конец работы функции second_func 2')  
8  
9  
10 def first_func():  
11     print('Начало работы функции first_func 3')  
12     second_func()  
13     print('Конец работы функции first_func 4')  
14  
15  
16 print(1)  
17 print(2)  
18 first_func()  
19 print(3)
```

Powered by  trinket

```
1  
2  
Начало работы функции first_func 3  
Начало работы функции second_func 1  
Внимание! Обработано исключение!  
Конец работы функции second_func 2  
Конец работы функции first_func 4  
3
```


Обработка исключений основной управляющей функцией

< >

main.py

+ ↗

```
1 def first_func():
2     print('Начало работы функции first_func 1')
3     try:
4         second_func()
5     except Exception as ex:
6         print(f'Внимание! Обработано исключение: {ex}')
7     print('Конец работы функции first_func 2')
8
9
10 def second_func():
11     print('Начало работы функции second_func 3')
12     third_func()
13     print('Конец работы функции second_func 4')
14
15
16 def third_func():
17     print('Начало работы функции third_func 5')
18     1 / 0
19     print('Конец работы функции third_func 6')
20
21
22 print(1)
23 print(2)
24 first_func()
25 print(3)
```

Powered by  trinket

1
2
Начало работы функции first_func 1
Начало работы функции second_func 3
Начало работы функции third_func 5
Внимание! Обработано исключение: division by zero
Конец работы функции first_func 2
3

Решение



```
< > main.py + ↗
1 def first_func():
2     print('Начало работы функции first_func 1')
3     try:
4         second_func()
5     except Exception as ex:
6         print(f'Внимание! Обработано исключение: {ex} на уровне first_func 1')
7     print('Конец работы функции first_func 2')
8
9
10 def second_func():
11     print('Начало работы функции second_func 3')
12     try:
13         third_func()
14     except Exception as ex:
15         print(f'Внимание! Обработано исключение: {ex} на уровне second_func 2')
16     print('Конец работы функции second_func 4')
17
18
19 def third_func():
20     print('Начало работы функции third_func 5')
21     try:
22         1 / 0
23     except Exception as ex:
24         print(f'Внимание! Обработано исключение: {ex} на уровне third_func 3')
25     print('Конец работы функции third_func 6')
26
27
28 print(1)
29 print(2)
30 first_func()
31 print(3)
```

Powered by  trinket

```
1
2
Начало работы функции first_func 1
Начало работы функции second_func 3
Начало работы функции third_func 5
Внимание! Обработано исключение: division by zero на уровне
third_func 3
Конец работы функции third_func 6
Конец работы функции second_func 4
Конец работы функции first_func 2
3

3
```

Инструкция raise

→

Инструкция raise

Инструкция raise позволяет **принудительно** вызвать исключение.

```
main.py
1 raise NameError('HiThere')
```

Powered by  trinket

```
Traceback (most recent call last):
  File "/tmp/sessions/93ae904939139f1b/main.py", line 1, in <module>
    raise NameError('HiThere')
NameError: HiThere
```

В общем случае инструкция raise повторно вызывает последнее исключение, которое было активным в текущей области видимости. Если нужно определить, было ли вызвано исключение, но не обрабатывать его, более простая форма инструкции raise позволяет повторно вызвать исключение.

```
main.py
1 try:
2     raise NameError('HiThere')
3 except NameError:
4     print('An exception flew by!')
5     raise
6
```

An exception flew by!

```
Traceback (most recent call last):
  File "/tmp/sessions/ff67b4c71fd4b226/main.py", line 2, in <module>
    raise NameError('HiThere')
NameError: HiThere
```

Предложение from используется для цепочки исключений. Если исключение задано, второе выражение должно быть другим классом или экземпляром исключения, который затем будет присоединен к брошенному исключению, который доступен для записи. Если возникшее исключение не обработано, то будут напечатаны оба исключения:

< >

main.py

+ ↗

1 ▾ try:

2 print(1 / 0)

3 ▾ except Exception as exc:

4 raise RuntimeError("Something bad happened") from exc

Powered by  trinket

Traceback (most recent call last):
 File "/tmp/sessions/757510ad4390c071/main.py", line 2, in <module>
 print(1 / 0)
ZeroDivisionError: division by zero

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
 File "/tmp/sessions/757510ad4390c071/main.py", line 4, in <module>
 raise RuntimeError("Something bad happened") from exc
RuntimeError: Something bad happened

Заключение →

Заключение (исключения)

- Стандартизированная обработка ошибок: используя встроенные исключения или создавая собственное исключение с более точным именем и описанием, вы можете адекватно определить событие ошибки, что поможет вам отловить событие, при котором данное исключение возникает;
- Более чистый код: исключения отделяют код обработки ошибок от обычного кода, что помогает нам легко поддерживать объемный код;
- Надежное приложение: с помощью исключений мы можем разработать надежное приложение, которое может эффективно обрабатывать ошибки;
- Распространение исключений: по умолчанию исключение распространяет стек вызовов, если вы его не перехватываете. Например, если какое-либо событие ошибки произошло во вложенной функции, вам не нужно явно перехватывать и пересылать его; автоматически он перенаправляется в вызывающую функцию, где вы можете его обработать;
- Различные типы ошибок: либо вы можете использовать встроенное исключение, либо создать свое собственное исключение и сгруппировать их по их обобщенному родительскому классу, либо дифференцировать ошибки по их фактическому классу

Срезы

```
main.py
1  #Строки
2  a = "abcdefg"
3  try:
4      print(a[2])#вывести символ по индексу
5      print(a[3:])#вывести символы после индекса
6      print(a[:4])#вывести символы до индекса
7      print(a[:4:2])#вывести символы с шагом до индекса
8      print(a[8])#вывести символ по индексу
9  except:
10     print("Неверный диапазон")
```

с
defg
abcd
ac
Неверный диапазон


Списки

Создать пустой список можно двумя способами:

- Использовать пустые квадратные скобки [];
- Использовать встроенную функцию, которая называется list.

< > main.py + ↕

```
1 #Списки:
2 num = "12345"
3 l1 = list(num)
4 print(l1, type(l1))
5 l2 = ["a", "b", "c", "d"]
6 print(l2, type(l2))
```


Powered by  trinket

```
['1', '2', '3', '4', '5'] <class 'list'>
['a', 'b', 'c', 'd'] <class 'list'>
```


Матрицы

< >

main.py

+ 


```
1 n, m = 2, 3
2
3 my_list = [[0] * m] * n
4 my_list[0][0] = 17
5
6 print(my_list)
```

Powered by  trinket


[[17, 0, 0], [17, 0, 0]]

< >

main.py

+ 

```
1 p, m = 4, 5
2
3 for i in range(1, n):
4     for j in range(1, m):
5         if j != m - 1:
6             print(str((i * j)).ljust(2), end=' ')
7         else:
8             print(str((i * j)), end='')
9     print()
```


Powered by 

1 2 3 4
2 4 6 8
3 6 9 12

Dataclass

Dataclass


```
main.py
1 class Customer:
2     def __init__(self, name, balance):
3         self.name = name
4         self.balance = balance
5
6     def __str__(self):
7         return f"Customer {self.name}, balance={self.balance}"
8
9
10 bob = Customer('bob', 100)
11 print(bob)
```

Powered by  Jupyter
Customer bob, balance=100


Dataclass призван автоматизировать генерацию кода в реализации

класс

```
main.py
1 from dataclasses import dataclass
2
3 @dataclass
4 class Customer:
5     name: str
6     balance: int
7
8
9 jack = Customer('jack', 500)
10 print(jack)
11 print(jack.name, jack.balance)
```

Powered by  Jupyter
Customer(name='jack', balance=500)
jack 500

Аннотация типов (определение поля)

```
> main.py +   
1 from dataclasses import dataclass  
2  
3  
4 @dataclass  
5 class Customer:  
6     name: str  
7     balance: int  
8  
9  
10 cust1 = Customer(32, 'hello')  
11 cust2 = Customer([2, 3, 'hello'], {4, 3, 2})  
12 print(cust1)  
13 print(cust2)
```

Powered by  trinket


```
Customer(name=32, balance='hello')  
Customer(name=[2, 3, 'hello'], balance={2, 3, 4})
```

- name: str говорит, что имя должно быть текстовой строкой (тип str).
- balance: int говорит, что количество должно быть целым числом (тип int).

Без указания типа у вас не получится создать dataclass.

Значение по умолчанию

Добавление к полям значений по умолчанию.

 main.py  

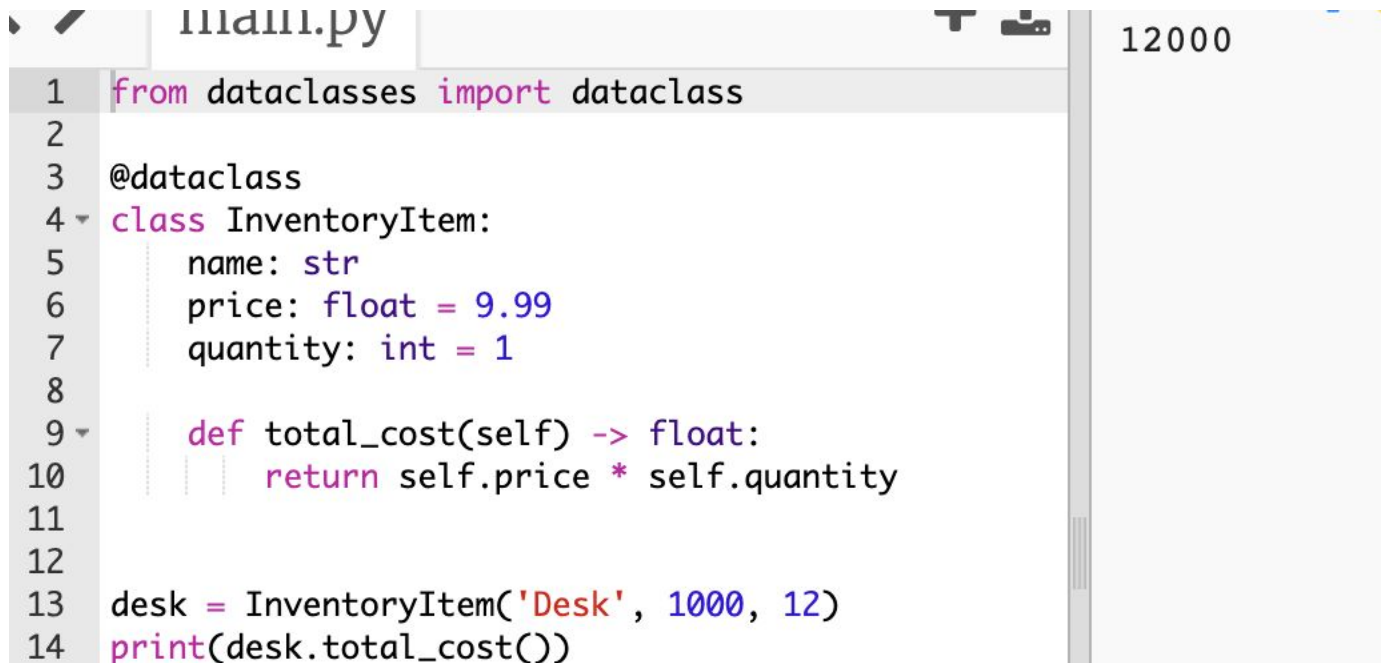
```
1 from dataclasses import dataclass
2
3 @dataclass
4 class InventoryItem:
5     name: str
6     price: float = 9.99
7     quantity: int = 1
8
9
10 desk = InventoryItem('Computer desk', 1000, 12)
11 pen = InventoryItem('Pen')
12 monitor = InventoryItem('Monitor', 300)
13 clock = InventoryItem('Clock', quantity=10)
14 print(desk)
15 print(pen)
16 print(monitor)
17 print(clock)
```

Powered by  trinket

```
InventoryItem(name='Computer desk', price=1000, quantity=12)
InventoryItem(name='Pen', price=9.99, quantity=1)
InventoryItem(name='Monitor', price=300, quantity=1)
InventoryItem(name='Clock', price=9.99, quantity=10)
```

Добавление методов

Как и с обычными классами вы можете свободно добавлять свои собственные методы в dataclass.




```
1 from dataclasses import dataclass
2
3 @dataclass
4 class InventoryItem:
5     name: str
6     price: float = 9.99
7     quantity: int = 1
8
9     def total_cost(self) -> float:
10         return self.price * self.quantity
11
12
13 desk = InventoryItem('Desk', 1000, 12)
14 print(desk.total_cost())
```

12000

Параметры dataclass

- 1.init: по умолчанию принимает значение True.
- 2.repr: по умолчанию принимает значение True.
- 3.eq : по умолчанию принимает значение True.
- 4.order: по умолчанию принимает значение False .
- 5.frozen: по умолчанию принимает значение False

```
< > main.py + ↕
1 from dataclasses import dataclass
2
3 @dataclass
4 class Point:
5     x: int
6     y: int
7
8 point1 = Point(5, 7)
9 point2 = Point(-10, 12)
10 point3 = Point(5, 7)
11
12 print(point1 == point2)
13 print(point1 == point3)
14 print(point2 != point3)
15 print(repr(point1))
```

Powered by  trinket

```
False
True
True
Point(x=5, y=7)
```

Изменяемость и неизменяемость атрибутов (frozen)

```
main.py
1 from dataclasses import dataclass
2
3
4 @dataclass
5 class InventoryItem:
6     name: str
7     price: float = 9.99
8     quantity: int = 1
9
10 desk = InventoryItem('Desk', 1000, 12)
11 print(desk)
12 desk.price = 800
13 desk.quantity = 5
14 print(desk)
```

```
InventoryItem(name='Desk', price=1000, quantity=12)
InventoryItem(name='Desk', price=800, quantity=5)
```

```
main.py
1 from dataclasses import dataclass
2
3
4 @dataclass(frozen=True)
5 class InventoryItem:
6     name: str
7     price: float = 9.99
8     quantity: int = 1
9
10 desk = InventoryItem('Desk', 1000, 12)
11 print(desk)
12 desk.price = 800
13 print(desk)
```

```
InventoryItem(name='Desk', price=1000, quantity=12)

Traceback (most recent call last):
  File "/tmp/sessions/bfc39204baaad914/main.py", line 12, in <module>
    desk.price = 800
  File "<string>", line 4, in __setattr__
dataclasses.FrozenInstanceError: cannot assign to field 'price'
```

order = False/True

```
main.py
1 from dataclasses import dataclass
2
3
4 @dataclass
5 class User:
6     name: str
7     age: int
8     rating: float
9
10
11 bob = User('Bob', 18, 77)
12 ash = User('Ash', 25, 100)
13 print(ash > bob)
```

Traceback (most recent call last):

File "/tmp/sessions/99c39135a032eaa8/main.py", line 13, in <module>

print(ash > bob)

TypeError: '>' not supported between instances of 'User' and 'User'

```
main.py
1 from dataclasses import dataclass
2
3
4 @dataclass(order=True)
5 class User:
6     name: str
7     age: int
8     rating: float
9
10
11 bob = User('Bob', 18, 77)
12 ash = User('Ash', 25, 100)
13 print(ash > bob)
```

Powered by
False

Особенность order в dataclass

Есть особенность использования `order=True`. Она заключается в том, что при сравнении будут использоваться все атрибуты класса, что не всегда полезно. Автоматически сгенерированные методы сравнения будут сравнивать следующие кортежи экземпляров между собой:

```
(self.name, self.age, self.rating)
```

А кортежи сравниваются по элементам.


Решение

< >

main.py

+ ↗

```
1 from dataclasses import dataclass, field
2
3
4 @dataclass(order=True)
5 class User:
6     name: str = field(compare=False)
7     age: int
8     rating: float
9
10
11 bob = User('Bob', 18, 77)
12 ash = User('Ash', 25, 100)
13 ken = User('Ken', 25, 120)
14 wolter = User('Wolter', 24, 120)
15 print(ash > bob)
16 print(ash > ken)
17 print(ash > wolter)
18
```

Powered by  trinket

True
False
True

При помощи `field(compare=False)` мы исключает атрибут `name` из кортежа сравнений.