

Лекция №6.

Заключительная

Зачет (автоматом)

- Лабораторные;
- Ответы на формы по лекции;
- Посещение.

Обработка исключений. Блоки `finally` и `else`

Необязательный блок `else`, который выполняется при штатном выполнении кода внутри блока `try`, то есть, когда не произошло никаких ошибок.

Необязательный блок `finally`, который, наоборот, выполняется всегда после блока `try`, вне зависимости произошла ошибка или нет.

Блок else

```
main.py
1 try:
2     x = int(input("x = "))
3     y = int(input("y = "))
4     res = x / y
5 except ZeroDivisionError as z:
6     print(z)
7 except ValueError as z:
8     print(z)
9 else:
10    print("Try - выполнено. Исключение не произошло.")
```

```
x = 4
y = 0
division by zero
```

```
main.py
1 try:
2     x = int(input("x = "))
3     y = int(input("y = "))
4     res = x / y
5 except ZeroDivisionError as z:
6     print(z)
7 except ValueError as z:
8     print(z)
9 else:
10    print("Try - выполнено. Исключение не произошло.")
```

```
x = 12
y = 4
Try - выполнено. Исключение не произошло.
```

Блок finally

```
main.py
1 try:
2     x, y = map(int, input().split())
3     res = x / y
4 except ZeroDivisionError as z:
5     print(z)
6 finally:
7     print("Блок finally выполняется всегда")
```

5 0
division by zero
Блок finally выполняется всегда

```
main.py
1 def get_values():
2     try:
3         x, y = map(int, input().split())
4         return x, y
5     except ValueError as v:
6         print("ValueError - ", v)
7         return 0, 0
8     finally:
9         print("finally выполняется до return")
10
11
12 x, y = get_values()
13 print(x, y)
14
```

Powered by trinket
5 a
ValueError - invalid literal for int() with base 10:
'a'
finally выполняется до return
0 0

Вложенные блоки try / except

```
main.py
1 try:
2     x, y = map(int, input().split())
3     try:
4         res = x / y
5         print("x / y = ", res)
6     except ZeroDivisionError:
7         print("Деление на ноль")
8 except ValueError as z:
9     print("Ошибка ValueError")
```

4 3
x / y = 1.3333333333333333

```
main.py
1 try:
2     x, y = map(int, input().split())
3     try:
4         res = x / y
5         print("x / y = ", res)
6     except ZeroDivisionError:
7         print("Деление на ноль")
8 except ValueError as z:
9     print("Ошибка ValueError")
```

3 0
Деление на ноль

Распространение исключений (propagation exceptions)


```
main.py
1 def func1():
2     return 1/0
3
4 func1()
```

Traceback (most recent call last):
File "/tmp/sessions/08202949adc09461/main.py", line 4, in <module>
func1()
File "/tmp/sessions/08202949adc09461/main.py", line 2, in func1
return 1/0
ZeroDivisionError: division by zero

```
main.py
1 def func2():
2     return 1/0
3
4 def func1():
5     return func2()
6
7 func1()
```

Traceback (most recent call last):
File "/tmp/sessions/99ea4582568ab76f/main.py", line 7, in <module>
func1()
File "/tmp/sessions/99ea4582568ab76f/main.py", line 5, in func1
return func2()
File "/tmp/sessions/99ea4582568ab76f/main.py", line 2, in func2
return 1/0
ZeroDivisionError: division by zero

```
main.py
1 def func2():
2     return 1/0
3
4 def func1():
5     return func2()
6
7 try:
8     print(func1())
9 except:
10    print("Error for func1")
```

Powered by  UMLike
Error for func1

Исключение, зародившееся на одном из уровней стека вызова, постепенно поднимается на самый верх. Это называется распространением исключений или propagation

```
main.py
1 def func2():
2     try:
3         return 1/0
4     except:
5         return "-- деление на ноль --"
6
7 def func1():
8     return func2()
9
10 try:
11     print(func1())
12 except:
13     print("Error for func1")
```

-- деление на ноль --


Функция isinstance()

main.py


```
1 class Geom(object):
2     pass
3 class Line:
4     pass
5 print(Geom.__class__)
6 print(Line.__class__)
```

main.py

```
1 class Geom(object):
2     pass
3
4
5 class Line(Geom):
6     pass
7
8 l = Line()
9 print(l.__class__)
10
11 print(isinstance(Line, Geom))
12 print(isinstance(Geom, Line))
13 print(isinstance(Line, object))
14
15 try:
16     print(isinstance(l, Geom))
17     print(isinstance(l, Line))
18 except:
19     print("Объект должен быть классом")
```

Powered by  Jupyter

<class 'type'>
<class 'type'>

Powered by  Jupyter

<class '__main__.Line'>
True
False
True
Объект должен быть классом


Наследование от встроенных типов данных

Лаб 7.4 Создайте класс `NewInt`, который унаследован от целого типа `int`, то есть произведем наследование поведения целых чисел и значит экземпляры нашего класса будут поддерживать те же операции, что и целые числа.

int
float
list
dict
tuple
set
...

} классы

```
> main.py
1 class Vector(list):
2     def __str__(self):
3         return " ".join(map(str, self))
4
5 v = Vector([1, 2, 3])
6 print(v)
7 v.append(123)
8 print(v)
9
10 print(v.__class__)
11 print(issubclass(Vector, list))
12 print(isinstance(v, list))
```

Powered by  trinket

```
1 2 3
1 2 3 123
<class '__main__.Vector'>
True
True
```

Полиморфизм и абстрактные методы

Полиморфизм – это возможность работы с совершенно разными объектами (языка Python) единым образом.

```
main.py
1 class Rectangle:
2     def __init__(self, w, h):
3         self.w = w
4         self.h = h
5     def get_rect_pr(self):
6         return 2*(self.w+self.h)
7 class Square:
8     def __init__(self, a):
9         self.a = a
10    def get_sq_pr(self):
11        return 4*self.a
12 r1 = Rectangle(1, 2)
13 r2 = Rectangle(3, 4)
14 print(r1.get_rect_pr(), r2.get_rect_pr())
15
16 s1 = Square(10)
17 s2 = Square(20)
18 print(s1.get_sq_pr(), s2.get_sq_pr())
19 geom = [r1, r2, s1, s2]
20 try:
21     for g in geom:
22         print(g.get_rect_pr())
23 except AttributeError as x:
24     print(x)
```

6 14
40 80
6
14
'Square' object has no attribute 'get_rect_pr'

Абстрактные методы

В языках программирования методы, которые обязательно нужно переопределять в дочерних классах и которые не имеют своей собственной реализации

```
main.py
1 class Geom:
2     def get_pr(self):
3         raise NotImplementedError("В дочернем классе должен быть переопределен метод get_pr()")
4 class Rectangle(Geom):
5     def __init__(self, w, h):
6         self.w = w
7         self.h = h
8     def get_rect_pr(self):
9         return 2*(self.w+self.h)
10    def get_pr(self):
11        return self.w * self.h
12 class Square(Geom):
13     def __init__(self, a):
14         self.a = a
15     def get_sq_pr(self):
16         return 4*self.a
17 r1 = Rectangle(1, 2)
18 r2 = Rectangle(3, 4)
19 print(r1.get_rect_pr(), r2.get_rect_pr())
20
21 s1 = Square(10)
22 s2 = Square(20)
23 print(s1.get_sq_pr(), s2.get_sq_pr())
24 geom = [r1, r2, s1, s2]
25 try:
26     for g in geom:
27         print(g.get_pr() )
28 except AttributeError as x:
29     print(x)
```


```
6 14
40 80
2
12

Traceback (most recent call last):
  File "/tmp/sessions/08df29064e3f194d/main.py", line 27, in <module>
    print(g.get_pr() )
  File "/tmp/sessions/08df29064e3f194d/main.py", line 3, in get_pr
    raise NotImplementedError("В дочернем классе должен быть переопределен метод
get_pr()")
NotImplementedError: В дочернем классе должен быть переопределен метод get_pr()
```

Массивы

Массив - это структура данных, в которой хранятся значения **одного типа** (<https://pythonz.net/references/named/array/>).

```
< > main.py + ⬆️ 📄
1 from array import *
2
3 my_array = array("i", [1,2,3,4])
4
5 print(type(my_array))
6 print(my_array)
7 my_array.append(23)
8 print(my_array)
9 try:
10     my_array.append("ads")
11 except:
12     print(None)
```

Powered by  trinket

```
<class 'array.array'>
array('i', [1, 2, 3, 4])
array('i', [1, 2, 3, 4, 23])
None
```

i, I	знаковый/ беззнаковый int соответственно	int	2	
------	--	-----	---	--

Перегрузка операторов

Перегрузка операторов в Python – это возможность с помощью специальных методов в классах переопределять различные операторы языка.

Имена таких методов включают двойное подчеркивание спереди и сзади

Перегрузка операторов

- `__init__()` – конструктор объектов класса, вызывается при создании объектов;
- `__str__()` – преобразование объекта к строковому представлению, вызывается, когда объект передается функциям `print()` и `str()` ;
- `__setattr__()` – вызывается, когда атрибуту объекта выполняется присваивание.

Перегрузка операторов. Пример

```
main.py
1 class Changeable:
2     def __init__(self, color):
3         self.color = color
4     def __call__(self, newcolor):
5         self.color = newcolor
6     def __str__(self):
7         return self.color
8
9 canvas = Changeable("green")
10 frame = Changeable("blue")
11 canvas("red")
12 frame("yellow")
13 print(canvas, frame)
```

red yellow

Визуализация действий в ЯП

Ссылка: <https://pythontutor.com/visualize.html>

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
([known limitations](#))

```
1 class Vector:
2     MIN_COORD = 0
3     MAX_COORD = 100
4
5     @classmethod
6     def validate(Vector, arg):
7         return Vector.MIN_COORD <= arg <= Vector.M
8
9 res = Vector.validate(5)
10 print(res)
```

Print output (drag lower right corner to resize)

True

Frames

Global frame

Vector	
res	True

Objects

Vector class

MAX_COORD	100
MIN_COORD	0
validate	classmethod instance