

Лекция №4.

Полиморфизм. Наследование.

Полиморфизм


Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Полиморфизм в операциях

Для целочисленных типов данных оператор «+» используется для выполнения **арифметической операции сложения**.

Подобным образом оператор «+» в строках используется для **конкатенации**.

```
< > main.py + ↕  
1 print("_____Конкатенация строк_____")  
2 str1 = "Give me "  
3 str2 = "Five"  
4 print(str1+str2)  
5 print("\n_____Арифметическая операция_____")  
6 num1 = 1  
7 num2 = 2  
8 print(num1 + num2)
```

Powered by  trinket

_____Конкатенация строк_____

Give me Five


_____Арифметическая операция_____

3

Полиморфные функции

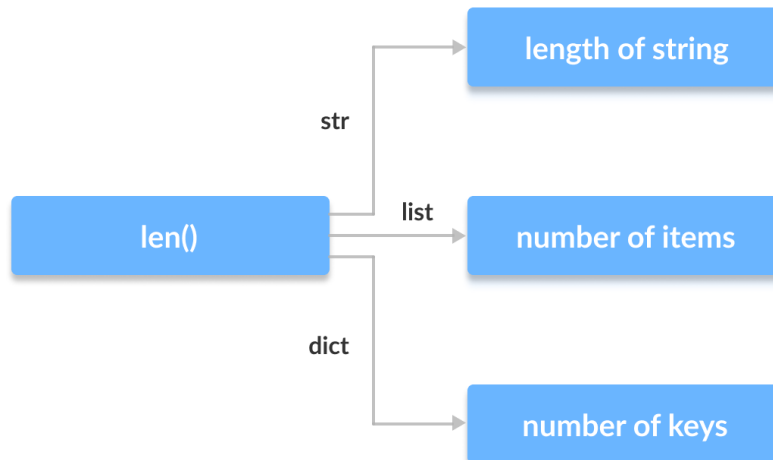
Различные типы данных, такие как строка, список, кортеж, множество и словарь могут работать с функцией `len()`. Однако, мы можем увидеть, что она возвращает специфичную для каждого типа данных информацию.

```
< > main.py + ↕
1 print("_____Полиморфные функции_____")
2 print(f'{len("Fizz Buzz")} - функция len для строки')
3 print(f'{len([1, 2, 3])} - функция len для списка')
4 print(f'{len({"Name": "John", "Address": "UK"})} - функция len для словаря')
5 |
```

Powered by  trinket

_____Полиморфные функции_____

9 - функция len для строки
3 - функция len для списка
2 - функция len для словаря



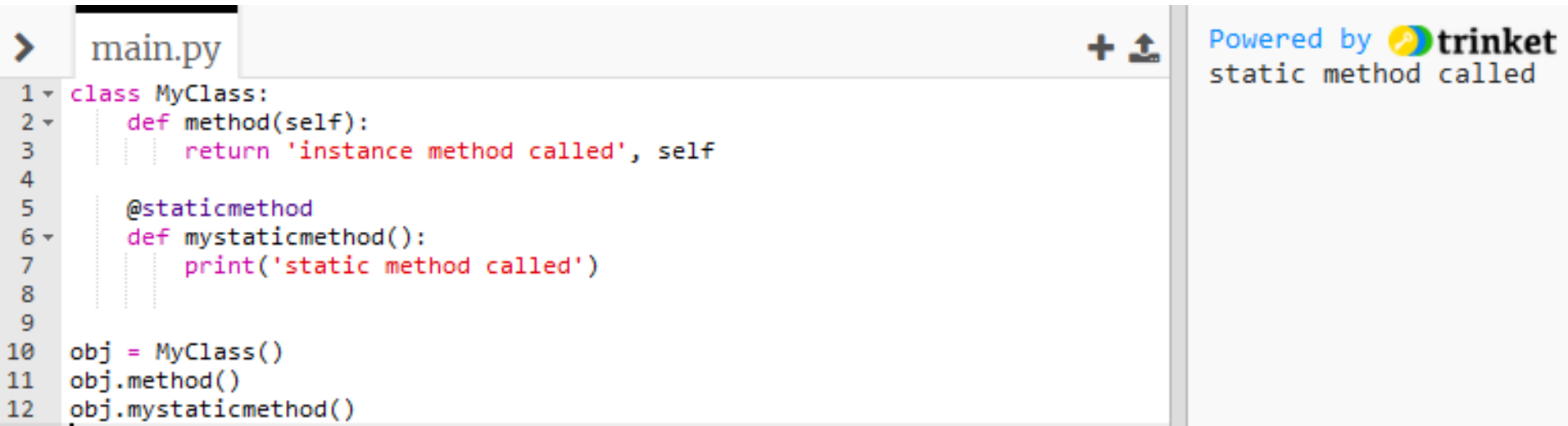
Полиморфизм в методах класса



```
< > main.py + ↕  
1 class Cat:  
2     def __init__(self, name):  
3         self.name = name  
4  
5     def info(self):  
6         print(f"I am a cat. My name is {self.name}.")  
7  
8     def make_sound(self):  
9         print("Meow")  
10  
11  
12 class Dog:  
13     def __init__(self, name):  
14         self.name = name  
15  
16     def info(self):  
17         print(f"I am a dog. My name is {self.name}.")  
18  
19     def make_sound(self):  
20         print("Bark")  
21  
22  
23 cat_obj = Cat("Ren")  
24 dog_obj = Dog("Stimpy")  
25  
26 for animal in (cat_obj, dog_obj):  
27     print(animal)  
28     animal.make_sound()  
29     animal.info()  
30     animal.make_sound()  
  
Powered by trinket  
<__main__.Cat object at 0x7f86ab39bfd0>  
Meow  
I am a cat. My name is Ren.  
Meow  
<__main__.Dog object at 0x7f86ab39be50>  
Bark  
I am a dog. My name is Stimpy.  
Bark
```

Статические методы в Python

Статические методы в Python – обычные функции(метод), помещенные в класс для удобства и находящиеся в пространстве имен этого класса. Это может быть какой-то вспомогательный код.

Обозначается при помощи **@staticmethod** в строке перед методом.




```
> main.py +  Powered by  trinket  
static method called  
1 class MyClass:  
2     def method(self):  
3         return 'instance method called', self  
4  
5     @staticmethod  
6     def mystaticmethod():  
7         print('static method called')  
8  
9  
10 obj = MyClass()  
11 obj.method()  
12 obj.mystaticmethod()
```

Пример с передачей параметров

> main.py

+ ↕

Powered by  trinket

```
1 from math import pi
2
3 class Cylinder:
4     @staticmethod
5     def make_area(d, h):
6         circle = pi * d ** 2 / 4 #Круг
7         side = pi * d * h #Сторона
8         return f"#make_area - {round(circle*2 + side, 2)}"
9
10    def __init__(self, di, hi):
11        self.dia = di #Диаметр
12        self.h = hi #Высота
13        self.area = self.make_area(di, hi)
14
15
16 a = Cylinder(1, 2)
17 print(a.area)
18
19 print(a.make_area(2, 2))
```

#make_area - 7.85
#make_area - 18.85

main.py



Powered by trinket

2020-12-30

Неправильная дата или формат строки с датой
2021-1-1

Неправильная дата или формат строки с датой

```
1 class Date(object):
2     def __init__(self, day=0, month=0, year=0):
3         self.day = day
4         self.month = month
5         self.year = year
6
7     @classmethod
8     def from_string(cls, date_as_string):
9         day, month, year = map(int, date_as_string.split('.'))
10        date1 = cls(day, month, year)
11        return date1
12
13    @staticmethod
14    def is_date_valid(date_as_string):
15        if date_as_string.count('.') == 2:
16            day, month, year = map(int, date_as_string.split('.'))
17            return day <= 31 and month <= 12 and year <= 3999
18
19    def string_to_db(self):
20        return f'{self.year}-{self.month}-{self.day}'
21
22    dates = [
23        '30.12.2020',
24        '30-12-2020',
25        '01.01.2021',
26        '12.31.2020'
27    ]
28
29    for string_date in dates:
30        if Date.is_date_valid(string_date):
31            date = Date.from_string(string_date)
32            string_to_db = date.string_to_db()
33            print(string_to_db)
34        else:
35            print(f'Неправильная дата или формат строки с датой')
```


Наследование в ООП

Наследование — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствуемой функциональностью.

Класс, от которого производится наследование, называется **базовым** или **родительским**. Новый класс — **потомком**, **наследником** или **производным** классом.


Использование наследования

< >

main.py

+ ↕

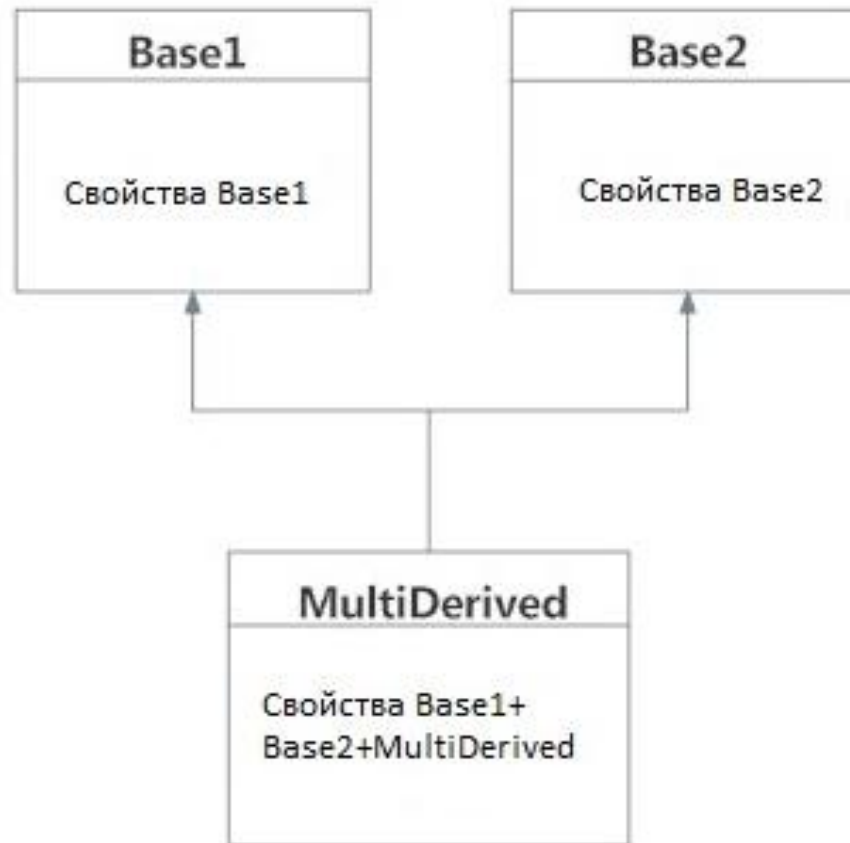
```
1 class Counter:
2     def __init__(self):
3         self.value = 0
4
5     def inc(self):
6         self.value += 2
7
8     def dec(self):
9         self.value -= 1
10
11 # Наследник Counter
12 class NonDecreasingCounter(Counter): # в скобках указан класс-предок
13     def __str__(self):
14         return f"{self.value} - NonDecreasingCounter"
15
16 n = NonDecreasingCounter()
17 for _ in range(3):
18     n.inc()
19     print(f"{n} - прибавили 2")
20     if n.value%3==1:
21         n.dec()
22         print(f"{n} - вычли 1")
```

Powered by  trinket

2 - NonDecreasingCounter - прибавили 2
4 - NonDecreasingCounter - прибавили 2
3 - NonDecreasingCounter - вычли 1
5 - NonDecreasingCounter - прибавили 2

Множественное наследование

Множественное наследование – это возможность класса иметь более одного родительского класса.




Множественное наследование

>

main.py

+ ↗

```
1  # класс работника
2  class Employee:
3      def work(self):
4          print("Employee works")
5
6  # класс студента
7  class Student:
8      def study(self):
9          print("Student studies")
10
11 # Наследование от классов Employee и Student
12 class WorkingStudent(Employee, Student):
13     pass
14
15 # класс работающего студента
16 tom = WorkingStudent()
17 tom.work()      # Employee works
18 tom.study()     # Student studies
```

Powered by  trinket

Employee works
Student studies

Многоуровневое наследование

В многоуровневом наследовании свойства родительского класса и наследуемого от него класса передаются новому наследуемому классу.


```
> main.py
1 class Base:
2     pass
3
4 class Derived1(Base):
5     pass
6
7 class Derived2(Derived1):
8     pass
9
```



Порядок разрешения методов (MRO)

Все классы в Python наследуются от класса object. Это базовый класс языка.

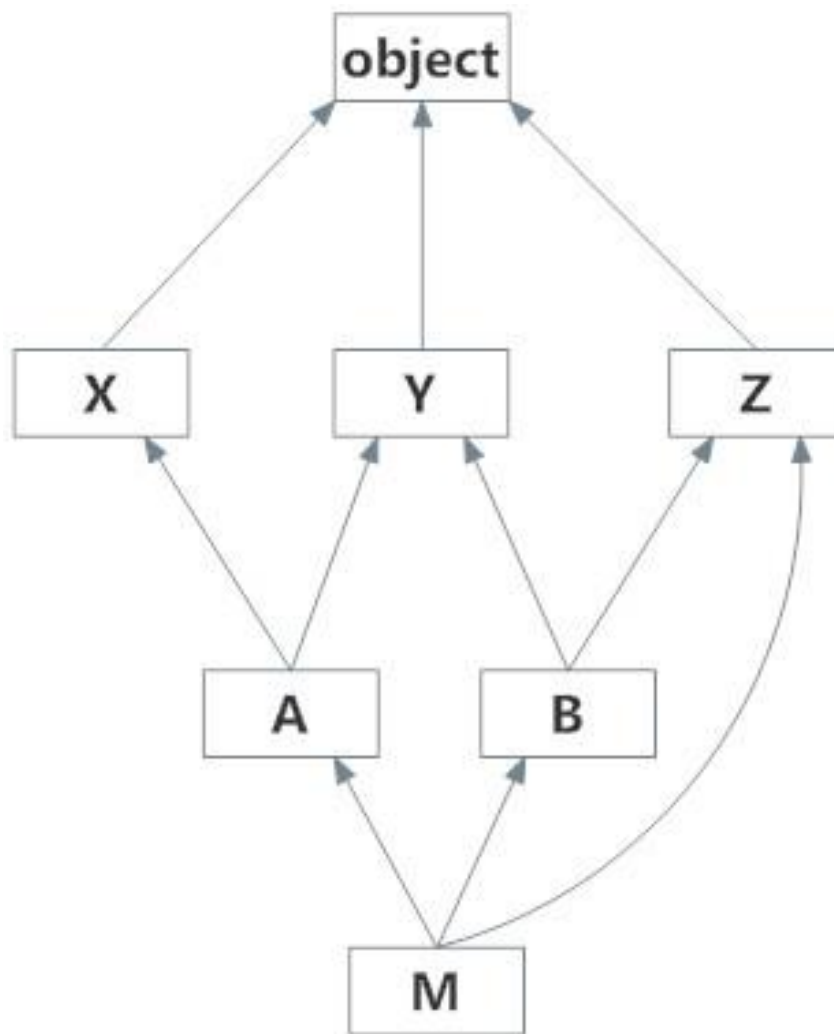
```
main.py
1 class X:
2     pass
3
4 class Y:
5     pass
6
7 class Z:
8     pass
9
10 class A(X, Y):
11     pass
12
13 class B(Y, Z):
14     pass
15
16 class M(B, A, Z):
17     pass
18
19 print("M - ", M.mro()) #Список
20 print("\nA - ", A.__mro__) #Кортеж
```

Powered by  trinket

M - [<class '__main__.M'>, <class '__main__.B'>, <class '__main__.A'>, <class '__main__.X'>, <class '__main__.Y'>, <class '__main__.Z'>, <class 'object'>]

A - (<class '__main__.A'>, <class '__main__.X'>, <class '__main__.Y'>, <class 'object'>)


Схема для вызова метода из класса



Неправильная линеаризация

> main.py + ↗


```
1 class X: ...
2 class Y(X): ...
3 class A(X, Y): ...
4
```

Powered by  trinket

Traceback (most recent call last):
 File "/tmp/sessions/747a3f7f4c57300c/main.py",
 line 3, in <module>
 class A(X, Y): ...
TypeError: Cannot create a consistent method
resolution
order (MRO) for bases X, Y

> main.py + ↗

```
1 class X: ...
2 class Y(X): ...
3 class A(Y, X): ...
4
```

Powered by  trinket

Базовый класс (способ реализации)

#1



main.py

```
1 class Person:
2     def __init__(self, name):
3         self.__name = name    # имя человека
4     def name(self):
5         return self.__name
6     def display_info(self):
7         print(f"Name: {self.__name} ")
8 class Employee(Person):
9     def work(self):
10        print(f"{self.name} works")
```

Переопределение метода #1

> main.py


```
1 class Person:
2     def __init__(self, name):
3         self.__name = name
4     def name(self):
5         return self.__name
6     def display_info(self):
7         print(f"Name: {self.__name}")
8 class Employee(Person):
9     def __init__(self, name, company):
10        super().__init__(name)
11        self.company = company
12    def display_info(self):
13        super().display_info()
14        print(f"Company: {self.company}")
15    def work(self):
16        print(f"{self.name} works")
17
18 tom = Employee("Tom", "Microsoft")
19 tom.display_info() # Name: Tom
20                  # Company: Microsoft
```

Powered by  trinket

Name: Tom
Company: Microsoft

Переопределение метода #2

```
> main.py + ↕  
1 from math import pi  
2 class Shape:  
3     def __init__(self, name):  
4         self.name = name  
5  
6     def area(self):  
7         pass  
8  
9     def fact(self):  
10        return "I am a two-dimensional shape."  
11  
12    def __str__(self):  
13        return self.name  
14 class Square(Shape):  
15     def __init__(self, length):  
16         super().__init__("Square")  
17         self.length = length  
18     def area(self):  
19         return self.length**2  
20     def fact(self):  
21         return "Squares have each angle equal to 90 degrees."  
22 class Circle(Shape):  
23     def __init__(self, radius):  
24         super().__init__("Circle")  
25         self.radius = radius  
26     def area(self):  
27         return pi*self.radius**2  
28 a = Square(4)  
29 b = Circle(7)  
30 print(b)  
31 print(b.fact())  
32 print(a.fact())  
33 print(b.area())
```

Powered by  trinket

Circle
I am a two-dimensional shape.
Squares have each angle equal to 90 degrees.
153.93804002589985

__slots__

```
main.py
1 class PointSlots:
2     ...
3     def __init__(self, x, y):
4         self.x = x
5         self.y = y
6     ...
7 p2 = PointSlots(3, 4)
8 p2.new_attr = 10
9 print(p2.__dict__)
```

Powered by  trinket
{'x': 3, 'y': 4, 'new_attr': 10}

```
main.py
1 class PointSlots:
2     __slots__ = ("x", "y")
3     def __init__(self, x, y):
4         self.x = x
5         self.y = y
6     ...
7 p2 = PointSlots(3, 4)
8 p2.new_attr = 10
9 print(p2.__dict__)
```

Powered by  trinket


Traceback (most recent call last):
 File "/tmp/sessions/a3769935dd30af80
/main.py", line 8, in <module>
 p2.new_attr = 10
AttributeError: 'PointSlots' object has no
attribute 'new_attr'

`__slots__` позволяет: ускорить работу программы и уменьшить размер выделяемой памяти

Использование памяти

Уменьшение количества занимаемой памяти при использовании `__slots__` связано с тем, что в `__slots__` хранятся только значения из пространства имён, а при использовании `__dict__` в память добавляется размер коллекции `__dict__` :

```
> main.py + ↕  
1 class Point:  
2     def __init__(self, x, y):  
3         self.x = x  
4         self.y = y  
5  
6 class PointSlots:  
7     __slots__ = ('x', 'y')  
8     def __init__(self, x, y):  
9         self.x = x  
10        self.y = y  
11  
12 s = Point(3, 4)  
13 print('No slots:', s.__sizeof__(), s.__dict__.__sizeof__()) # No slots: 32 88  
14 d = PointSlots(3, 4)  
15 print('Slots', d.__sizeof__()) # Slots 32
```

Powered by  trinket
No slots: 32 88
Slots 32