STEP IT ACADEMY

Занятие №15

ООП. Полиморфизм

Вспомним использование конструктора __init__

```
class Book:
        def__init_(self, name, author): self.name = name
                self.author = author

        def get_name(self):  return
                self.name


        def get_author(self):  return
                self.author
```

инициализатор

```
book = Book('Война и мир', 'Толстой Л. Н.')
print('{}, {}'.format(book.get_name(), book.get_author())) # Война и мир, Толстой Л.
Н.
```

Полиморфизм на примере +

Свойство кода работать с разными типами данных называют полиморфизмом.

Мы уже неоднократно пользовались этим свойством многих функций и операторов, не задумываясь о нѐм. Например, оператор + является полиморфным:

```
print(1 + 2)                    # 3
print(1.5 + 0.2)  print("abc"   # 1.7
+ "def")                        # abcdef
```

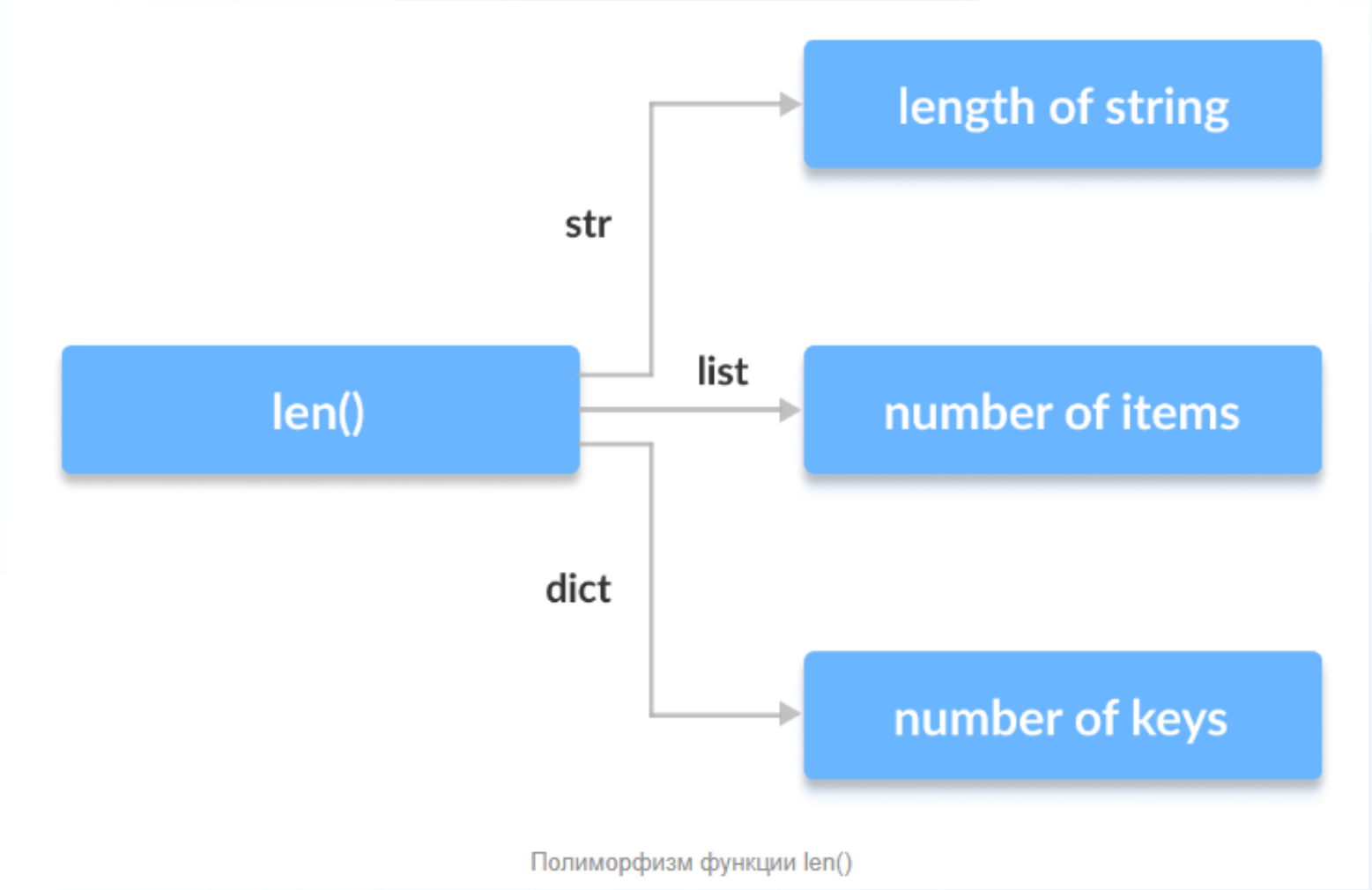
Усложним задачу

```
def f(x,          y):
        return x +          y


print(f(1, 2))                          # 3
print(f(1.5, 0.2))                      # 1.7
print(f("abc", "def"))                  # abcdef
```

Полиморфизм на примере функции len()

```
print(len("Programiz"))
print(len(["Python", "Java", "C"]))
print(len({"Name": "John", "Address": "Nepal"}))
```

Визуальное представление примера



Полиморфизм функции len()

Рассмотрим пример использования полиморфизма в классах

1 пример
Реализуем классы

```
class Cat:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print(f"I am a cat. My name is {self.name}. I am {self.age}
years old.")

    def make_sound(self):
        print("Meow")
```

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print(f"I am a dog. My name is {self.name}. I am {self.age}
years old.")

    def make_sound(self):
        print("Bark")
```

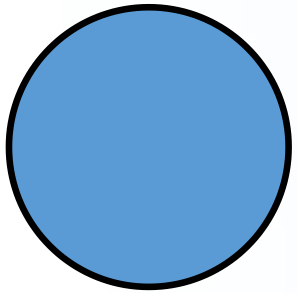
```
cat1 = Cat("Kitty", 2.5)
dog1 = Dog("Fluffy", 4)

for animal in (cat1, dog1):
    animal.make_sound()
    animal.info()
    animal.make_sound()
```

2 пример
Реализуем классы

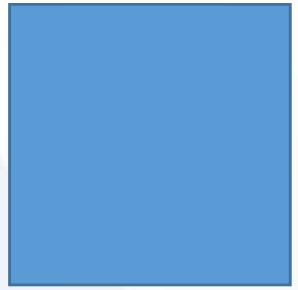
```
class T1:
    def __init__(self):
        self.n = 10

    def total(self, a):
        return self.n + int(a)


class T2:
    def __init__(self):
        self.string = 'Hi'

    def total(self, a):
        return len(self.string + str(a))
```

```
t1 = T1()
t2 = T2()

print(t1.total(35))  # Вывод: 45
print(t2.total(35))  # Вывод: 4
```

3 пример



- Площадь: πr^2
- Периметр: $2\pi r$



- Площадь: a^2
- Периметр: $4a$

Реализуем классы

```
from math import pi


class Circle:
    def _init_(self, radius): self.radius =
        radius

    def area(self):
        return pi * self.radius ** 2

    def perimeter(self):
        return 2 * pi * self.radius
```

```
from math import pi


class Square:
    def __init__(self, side): self.side
        = side

    def area(self):
        return self.side * self.side

    def perimeter(self):  return 4
        * self.side
```

Полиморфная функция

Теперь мы можем определить полиморфную функцию print_shape_info, которая будет печатать данные о фигуре:

```
def print_shape_info(shape):
        print("Area = {}, perimeter = {}.".format(shape.area(),
                                                  shape.perimeter()))


square = Square(10)
# Area = 100, perimeter = 40.
print_shape_info(square) circle =
Circle(10)
# Area = 314.15926535897793, perimeter = 62.83185307179586.
print_shape_info(circle)
```

Полиморфная функция

Если аргумент функции print_shape_info — экземпляр класса Square, то выполняются методы, определѐнные в этом классе, если экземпляр Circle, то выполняются методы Circle.

Выбор конкретной реализации вычисления площади и периметра производится в момент вызова методов area и perimeter и зависит от класса экземпляра.

Функция dir

```
print(dir(square))          # Свойства и спецметоды экземпляра.
print(dir(Square))          # Свойства и спецметоды класса.
```

Класс «Прямоугольник»

```
class Rectangle:
        def init(self, width, height): self.width   =
                width  self.height = height

        def area(self):
                return self.width * self.height

        def perimeter(self):
                return 2 * (self.width + self.height)


rect = Rectangle(10, 15)
print_shape_info(rect)                  # Area = 150, perimeter = 50.
```

Интерфейс

Возможность единообразной работы с экземплярами Square, Circle и Rectangle появилась потому, что эти классы предоставляют одинаковый интерфейс:

- одинаковые имена методов
- одинаковые параметры (в данном случае – только self)
- одинаковый смысл возвращаемых значений (в данном случае – числа)


STEP
IT ACADEMY