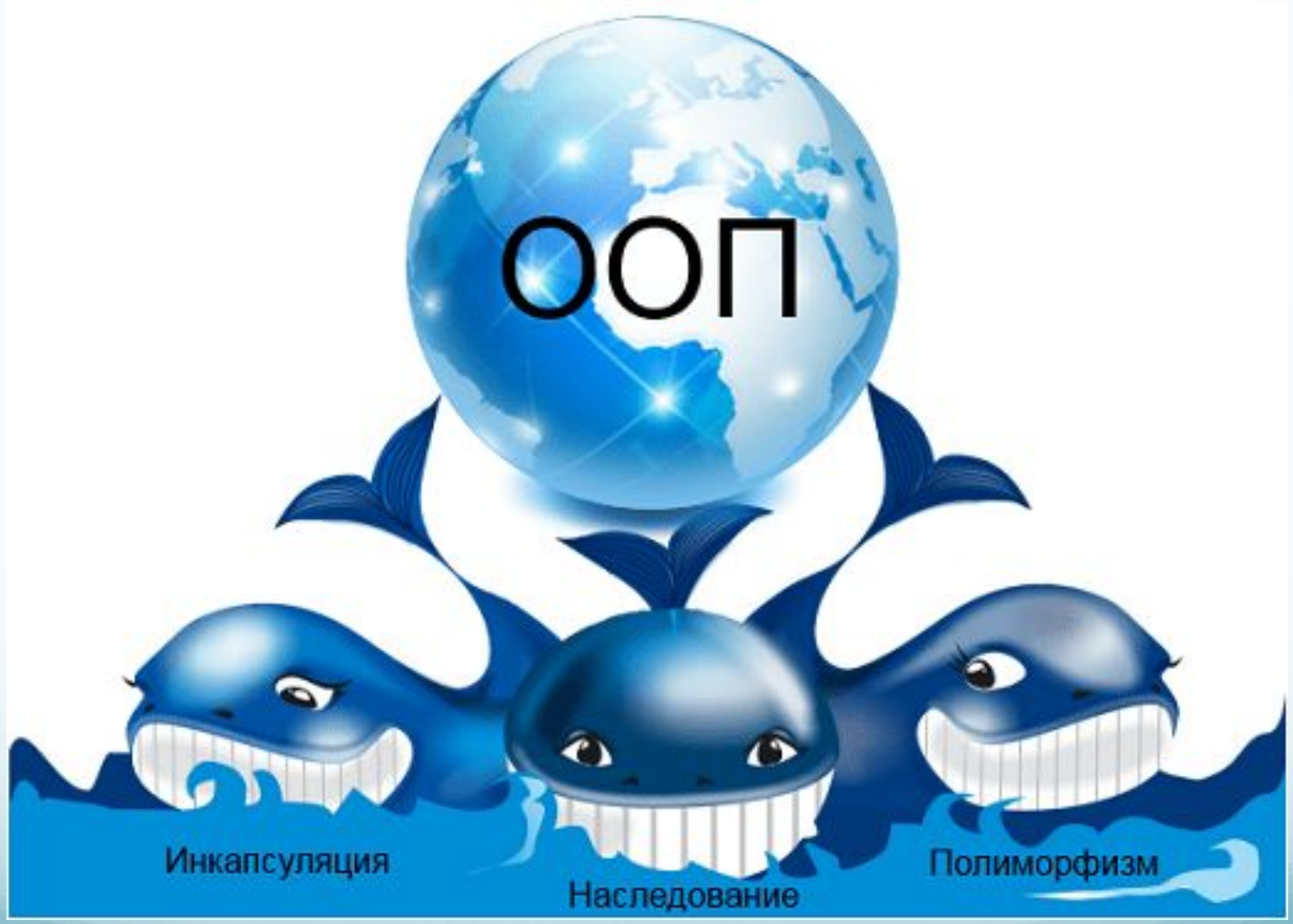
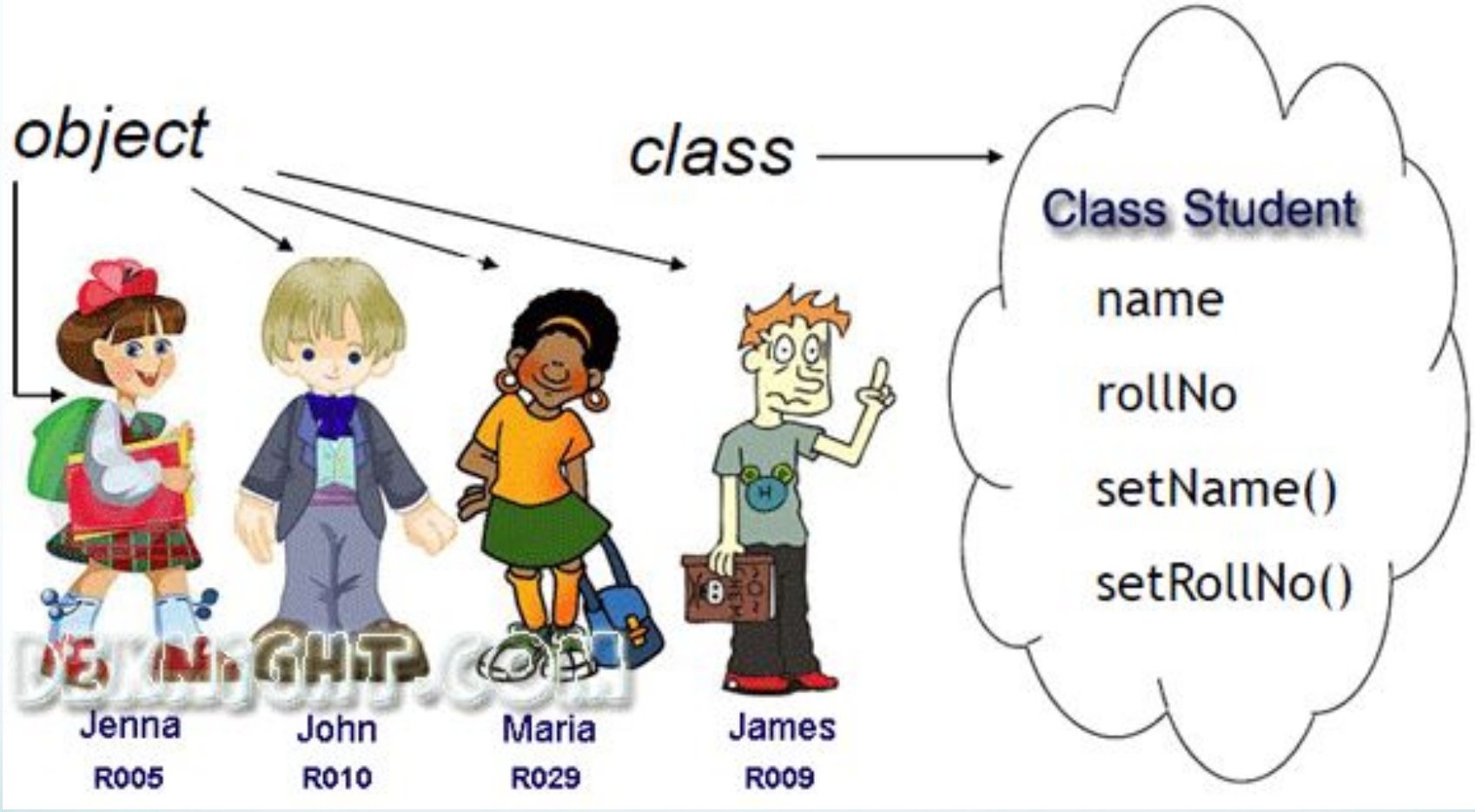
ООП

PYTHON


ООП
Инкапсуляция
Наследование
Полиморфизм


object
class
Class Student
name
rollNo
setName()
setRollNo()
Jenna
R005
John
R010
Maria
R029
James
R009


КЛАСС
ОБЪЕКТ
ОБЪЕКТ
ОБЪЕКТ
...
...
...
...
...
...
СВОЙСТВА
МЕТОДЫ
СОБЫТИЯ

```
class Address():
    name = ""
    city=""
    state=""
```

```
#В коде, приведённом выше, Address является именем класса.
#Переменные в классе, такие, как name и city, называются атрибутами или полями.
#В отличие от функции, имена классов должны начинаться с большой буквы.
```

```
# Создать инстанцию класса address
homeAddress = Address()

# Задать поля в address
homeAddress.name = "Natasha Burakova"
homeAddress.city = "Moscow"
homeAddress.state = "RU"
```

```
# Вывести адрес на экран
def printAddress(address):
    print (address.name)
    print( address.city+", "+address.state)


# Создать другой адресс
vacationHomeAddress = Address()

# Задать поля этого адреса
vacationHomeAddress.name = "Natasha Burakova"
vacationHomeAddress.city = "New yourk"
vacationHomeAddress.state = "UA"

printAddress( homeAddress )
print()
printAddress( vacationHomeAddress )
```

```python
#Помимо атрибутов,  у класса могут быть методы.  Метод - это функция,  которая существует внутри класса.

class Dog():
    age = 0
    name = ""
    weight = 0
    def bark(self):
        print("Woof says", self.name) #доступ к атрибуту имени осуществляется через переменную self.


myDog = Dog()

myDog.name = "Sharik"
myDog.weight = 20
myDog.age = 3

#Строка вызывает метод bark
myDog.bark() #несмотря на то, что у функции bark есть один параметр,  self, вызов не передаёт ей ничего.
```

```python
class Person:
    name = ""
    money = 0


bob = Person()
bob.name = "Bob"
bob.money = 100
# переменная bob не является объектом Person. Переменная bob - ссылка на объект Person.

nancy = bob
nancy.name = "Nancy"

print(bob.name, "has", bob.money, "dollars.")
print(nancy.name, "has", nancy.money, "dollars.")
```

```
def giveMoney1(money):
    money += 100


class Person:
    name = ""
    money = 0


bob = Person()
bob.name = "Bob"
bob.money = 100

giveMoney1(bob.money)
print(bob.money)


def giveMoney2(Person):
    Person.money += 100

giveMoney2(bob)
print(bob.money)
```

```
#Добавив функцию под названием constructor, программист может добавить код,
# автоматически вызываемый каждый раз во время создании инстанции класса.
#пример конструктора ниже:
class Dog():
    name = ""

    # Конструктор
    # Вызывается в момент создания объекта этого типа
    def __init__(self):
        print("A new dog is born!")



myDog = Dog()
```

```
class Dog():
    name = ""

    # Конструктор
    # Вызывается на момент создания объекта этого типа
    def __init__(self, newName):
        self.name = newName


# Это создаёт собаку
myDog = Dog("Spot")

# Вывести имя собаки, убедиться, что оно было установлено
print(myDog.name)

# Эта строка выдаст ошибку, потому что
# конструктору не было передано имя
herDog = Dog()
```

```
class Dog():
    name = "Rover"

    # Конструктор
    # Вызывается во время создания объекта
    def __init__(self, name):
        print(self.name)
        print(name)


# Это создаёт собаку
myDog = Dog("Spot")
```

```
#Должны ли имена класса начинаться с маленькой или большой буквы?
# Должны ли имена методов класса начинаться с маленькой или большой буквы?
# Что в классе следует перечислить сначала - атрибуты или методы?
# Как иначе можно назвать ссылку?
# Какое другое название переменной инстанции?
# Как иначе можно назвать инстанцию класса?
# Создайте класс под названием Star, который будет выводить "A star is born!" каждый раз во время его создания.
# Создайте класс Monster с атрибутами для здоровья и имени. Добавьте конструктор к этому классу,
# устанавливающий здоровья и имя равными значениям,  переданным как параметры.
```

```
# Наследование


class Person():
    name = ""


class Employee(Person):
    job_title = ""


class Customer(Person):
    email = ""


johnSmith = Person()
johnSmith.name = "John Smith"

janeEmployee = Employee()
janeEmployee.name = "Jane Employee"
janeEmployee.job_title = "Web Developer"

bobCustomer = Customer()
bobCustomer.name = "Bob Customer"
bobCustomer.email = "send_me@spam.com"
```

```
#Методы также наследуются.
# Нижеприведённый код выведет “Person created” три раза
class Person():
    name = ""

    def __init__(self):
        print("Person created")


class Employee(Person):
    job_title = ""


class Customer(Person):
    email = ""


johnSmith = Person()
janeEmployee = Employee()
bobCustomer = Customer()
```

```python
#В нижеприведённом коде, дочерние классы содержат свои конструкторы,
class Person():
    name = ""

    def __init__(self):
        print("Person created")


class Employee(Person):
    job_title = ""

    def __init__(self):
        print("Employee created")


class Customer(Person):
    email = ""

    def __init__(self):
        print("Customer created")


johnSmith = Person()
janeEmployee = Employee()
bobCustomer = Customer()
```

```python
#Если вы захотите вызвать метод как родительского,  так и дочернего класса,  дочерний класс может вызвать родительский метод:
class Person():
    name = ""

    def __init__(self):
        print("Person created")


class Employee(Person):
    job_title = ""

    def __init__(self):
        Person.__init__(self)
        print("Employee created")


class Customer(Person):
    email = ""

    def __init__(self):
        Person.__init__(self)
        print("Customer created")


johnSmith = Person()
janeEmployee = Employee()
bobCustomer = Customer()
```

```python
class Person: #super class
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def displayPerson(self):
        print(self.name, "is", self.age, "years old")
    def getName(self):
        print(self.name)
    def setName(self, name):
        self.name = name

class Student(Person): #subclass or child class
    def __init__(self, name, age, university):
        super().__init__(name, age)
        self.university = university
    def displayStudent(self):
        print(self.name, "is", self.age, "years old studies at", self.university)

natasha = Person("Natasha", 20)
natasha.displayPerson()
natasha.setName("Nata")
natasha.getName()

nata = Student("Nata", 20, "HSE")
nata.displayStudent()
```

```
# Задание
# Напишите код, описывающий класс Animal:
# Добавьте атрибут имени животного.
# Добавьте метод eat(), выводящий “Ням ням.”
# Добавьте метод makeNoise(), выводящий “[animal name] говорит Грр.”
# Добавьте конструктор классу Animal, выводящий “ Родилось животное.”
# Класс Cat:
# Пусть Animal будет его родительским классом.
# Метод makeNoise() класса Cat выводит “[animal name] говорит Мяу.”
# Конструктор для Cat выводит “Родился кот”, а также вызывает родительский конструктор.
# Класс Dog:
# Пусть Animal будет его родительским классом.
# Метод makeNoise() для Dog выводит “[animal name] говорит Гав.”
# Конструктор Dog выводит “ Родилась собака.”, а также вызывает родительский конструктор.
# Основная программа:
# Код, создающий кота, двух собак и одно простое животное.
# Дайте имя каждому животному.
# Код, вызывающий eat() и makeNoise() для каждого животного.
```