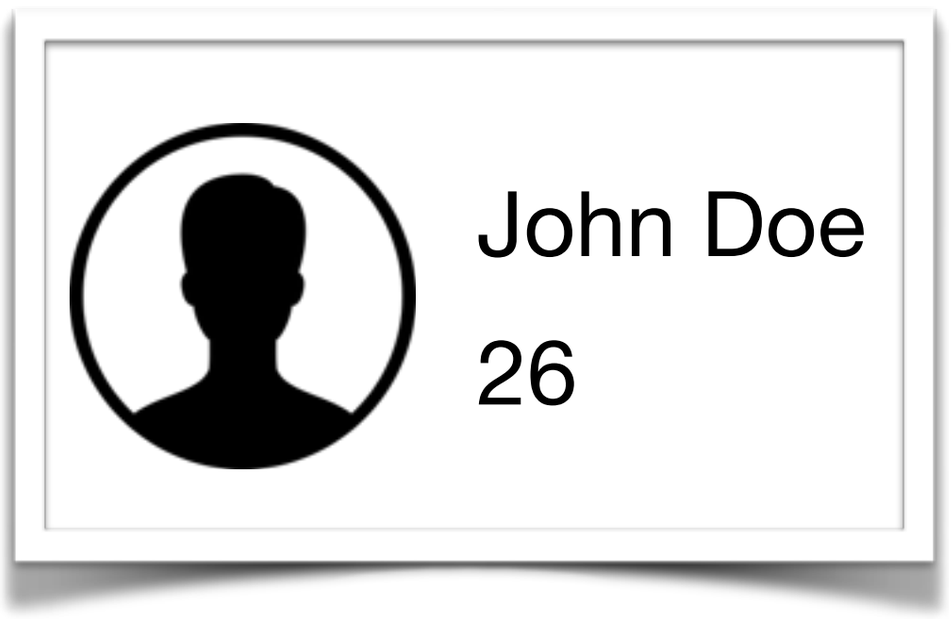
Python как первый язык

День 4

Проблема!

Сложные программы – требуют сложных данных

```
01: user = {}
02: user["name"] = "John Doe"
03: user["age"] = 26
```



Сложные программы – требуют сложных данных

```
01: user = {}
02: user[”name”] = ”John Doe”
03: user[”age”] = 26
```

```
01: user = {}
02: user[”name”] = 26
03: user[”age”] = ”John Doe”
```

```
01: user = {}
02: user[”name”] = []
03: user[”age”] = -666
```

Сложные программы – требуют сложных данных

```
01: user = {}
02: user["name"] = "John Doe"
03: user["age"] = 26
```

```
01: user = {}
02: user["name"] = 26
03: user["age"] = "John Doe"
```

```
01: user = {}
02: user["name"] = []
03: user["age"] = -666
```

Сложные программы – требуют сложных данных

```
01: def create_user(name, age):
02:     if type(name) != str:
03:         raise TypeError(”name is not string”)
04:
05:     if type(age) != int:
06:         raise TypeError(”age is not integer”)
07:
08:     return {
09:         ”name”: name,
10:         ”age”: age
11:     }
```

Сложные программы – требуют сложных данных

```
01: user = create_user(”John Doe”, 26)
```

Сложные программы – требуют сложных данных

```
01: user = create_user(”John Doe”, 26)
```

А как же методы?

Методы

```
01: def greet(user):
02:     print(”Hello, {0}”.format(user[”name”]))
```

Методы

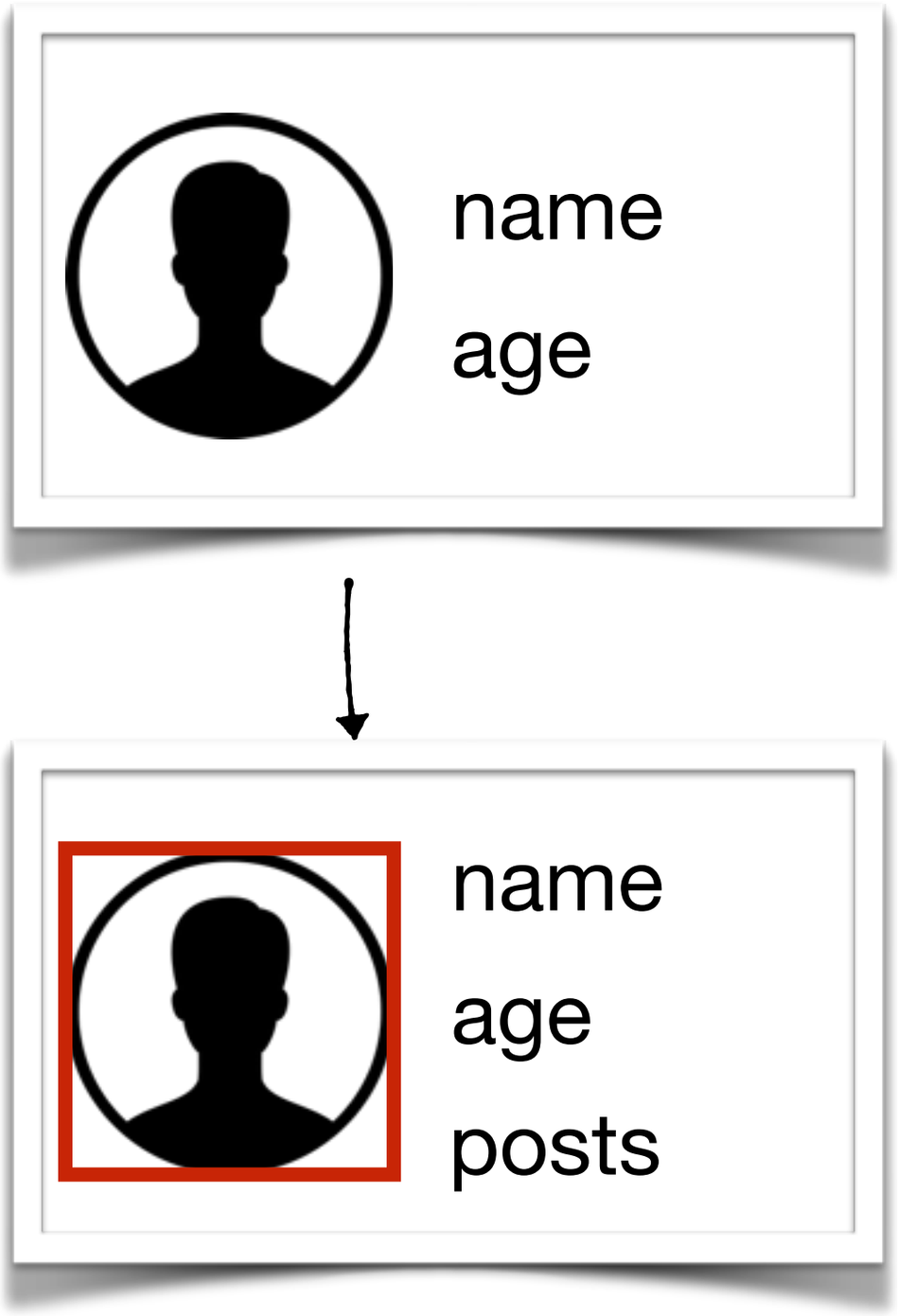
```
01: def greet(user):
02:     print(”Hello, {0}”.format(user[”name”]))
```

```
01: def rename(user, new_name):
02:     user[”name”] = new_name
02:     return user
```

Иерархия сущностей

Author

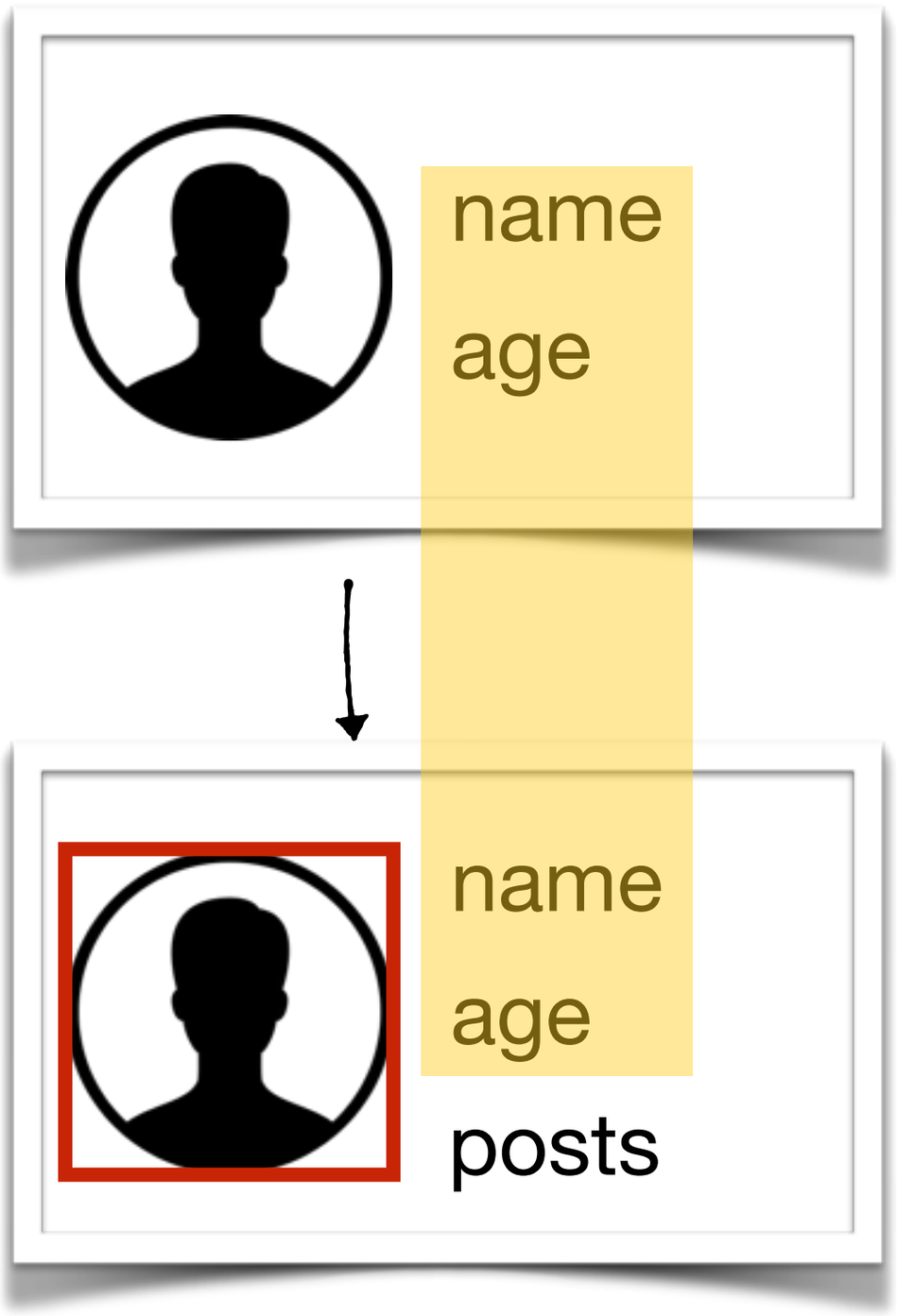
Иерархия сущностей



User

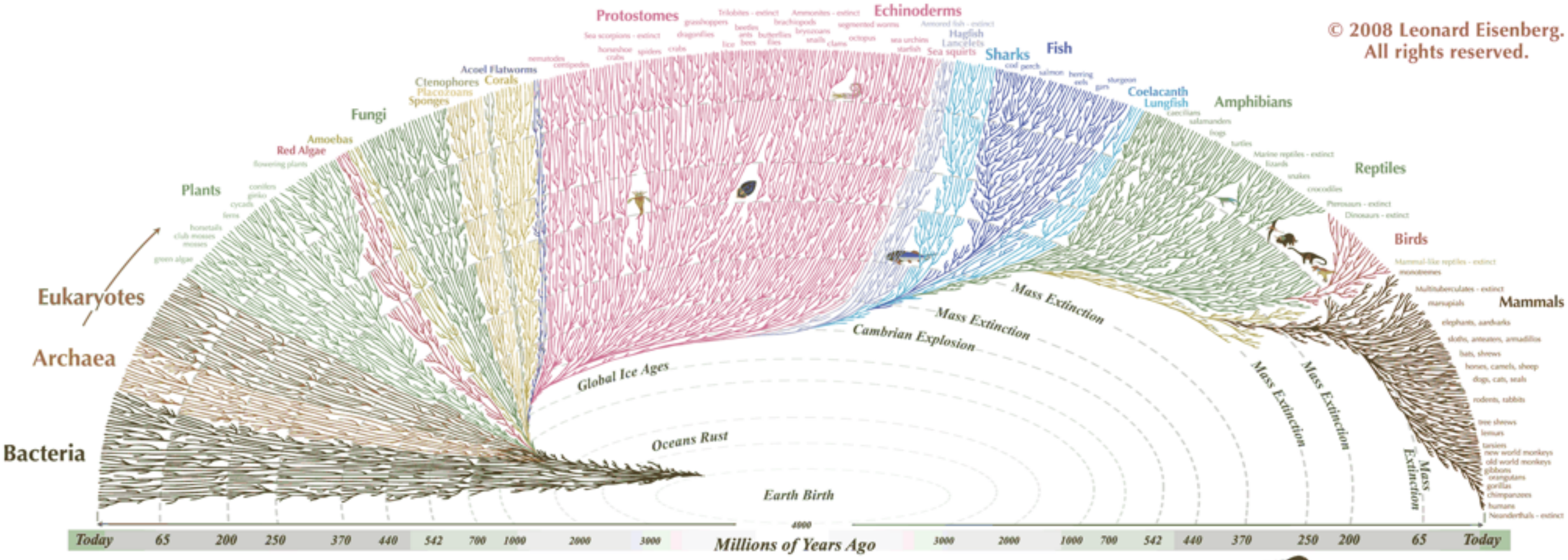
Author

Иерархия сущностей



User

Author



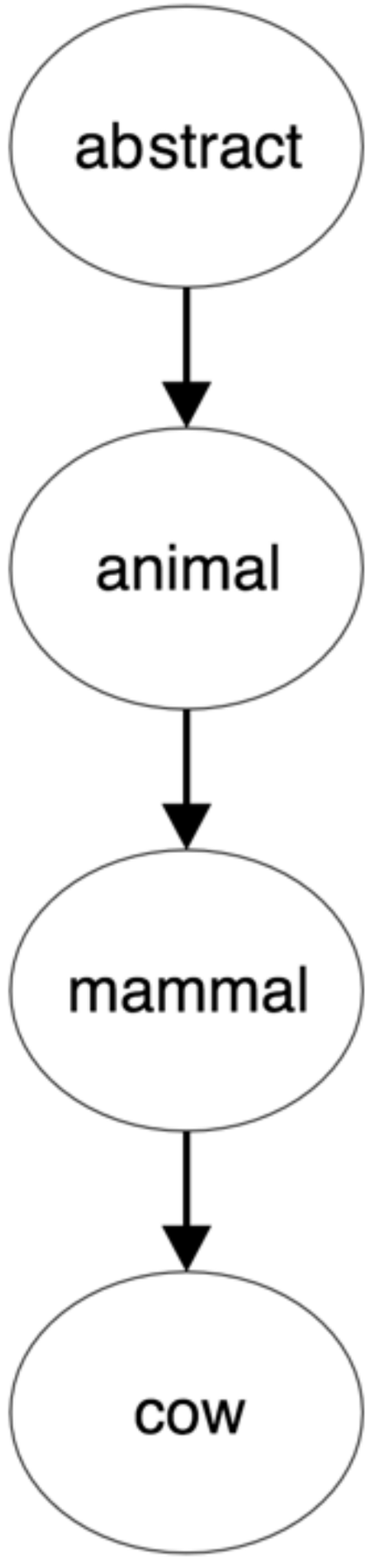
All the major and many of the minor living branches of life are shown on this diagram, but only a few of those that have gone extinct are shown. Example: Dinosaurs - extinct

Абстрактный класс существ

Класс всех животных

Класс млекопитающих

Класс коров



Создание классов

```
01: class User:
01:
02:     def __init__(self, name, age):
03:         self.name = name
04:         self.age = int(age)
05:
06:     def greet(self):
07:        print(”Hello! My name is {0}!”.format(self.name))
```

Создание классов

```
01: class User:
01:
02:     def __init__(self, name, age):
03:         self.name = name
04:         self.age = int(age)
05:
06:     def greet(self):
07:        print(”Hello! My name is {0}!”.format(self.name))
08:
09:
10: user = User(”John”, 22)
```

Создание классов

```
01: class User:
01:
02:     def __init__(self, name, age):
03:         self.name = name
04:         self.age = int(age)
05:
06:     def greet(self):
07:        print(”Hello! My name is {0}!”.format(self.name))
08:
09:
10: user = User(”John”, 22)
11:
12: print(user.name)
```

Создание классов

```
01: class User:
01:
02:     def __init__(self, name, age):
03:         self.name = name
04:         self.age = int(age)
05:
06:     def greet(self):
07:        print(”Hello! My name is {0}!”.format(self.name))
08:
09:
10: user = User(”John”, 22)
11:
12: print(user.name)
13:
12: user.greet()
```

Создание классов

```
01: class User:
01:
02:     def __init__(self, name, age):
03:         self.name = name
04:         self.age = int(age)
05:
06:     def greet(self):
07:        print(”Hello! My name is {0}!”.format(self.name))
```

Создание классов

```
01: class User:
01:
02:     def __init__(self, name, age):
03:         self.name = name
04:         self.age = int(age)
05:
06:     def greet(self):
07:        print(”Hello! My name is {0}!”.format(self.name))
```

Имя класса: User

Создание классов

```
01: class User:
01:
02:     def __init__(self, name, age):
03:         self.name = name
04:         self.age = int(age)
05:
06:     def greet(self):
07:        print(”Hello! My name is {0}!”.format(self.name))
```

Имя класса: User
Инициализируются с помощью свойств name и age

Создание классов

```
01: class User:
01:
02:     def __init__(self, name, age):
03:         self.name = name
04:         self.age = int(age)
05:
06:     def greet(self):
07:        print(”Hello! My name is {0}!”.format(self.name))
```

Имя класса: User
Инициализируются с помощью свойств name и age
self - ссылается на экземпляр класса

Создание классов

```
01: class User:
01:
02:     def __init__(self, name, age):
03:         self.name = name
04:         self.age = int(age)
05:
06:     def greet(self):
07:         print(”Hello! My name is {0}!”.format(self.name))
```

Имя класса: User
Инициализируются с помощью свойств name и age
self - ссылается на экземпляр класса
Функции становятся методами класса

Специальные методы

```
01: class User:
02:
03:     def __init__(self, name, age):
04:         self.name = name
05:         self.age = int(age)
06:
07:     def __lt__(self, other):
08:        return self.age < other.age
09:
10:     def __gt__(self, other):
11:        return self.age > other.age
```

Специальные методы

```
01: class User:
02:
03:     def __init__(self, name, age):
04:         self.name = name
05:         self.age = int(age)
06:
07:     def __lt__(self, other):
08:        return self.age < other.age
09:
10:     def __gt__(self, other):
11:        return self.age > other.age
```

```
01: user1 = User(”Gabe”, 22)
02: user2 = User(”Ikar”, 31)
03:
04: user1 > user2
05: user1 < user2
```

Специальные методы

```
01: class User:
02:
03:     def __init__(self, name, age):
04:         self.name = name
05:         self.age = int(age)
06:
07:     def __str__(self):
08:         return self.name
```

```
01: user = User(”Gabe”, 22)
02: print(user)
```

Наследование

```
01: class Author(User):
01:
02:     def __init__(self, name, age, posts=[]):
03:         super().__init__(name, age)
04:         self.posts = list(posts)
05:
06:     def publish(self, post):
07:         self.posts.append(post)
08:         print(post)
09:
10: admin = Author(”Jack”, 24)
```

Наследование

```
01: class Author(User):
01:
02:     def __init__(self, name, age, posts=[]):
03:         super().__init__(name, age)
04:         self.posts = list(posts)
05:
06:     def publish(self, post):
07:         self.posts.append(post)
08:         print(post)
09:
10: admin = Author(”Jack”, 24)
```

Класс - предок

Наследование

```
01: class Author(User):
01:
02:     def __init__(self, name, age, posts=[]):
03:         super().__init__(name, age)
04:         self.posts = list(posts)
05:
06:     def publish(self, post):
07:         self.posts.append(post)
08:         print(post)
09:
10: admin = Author(”Jack”, 24)
```

Класс - предок

super() - получение доступа к родительским методам

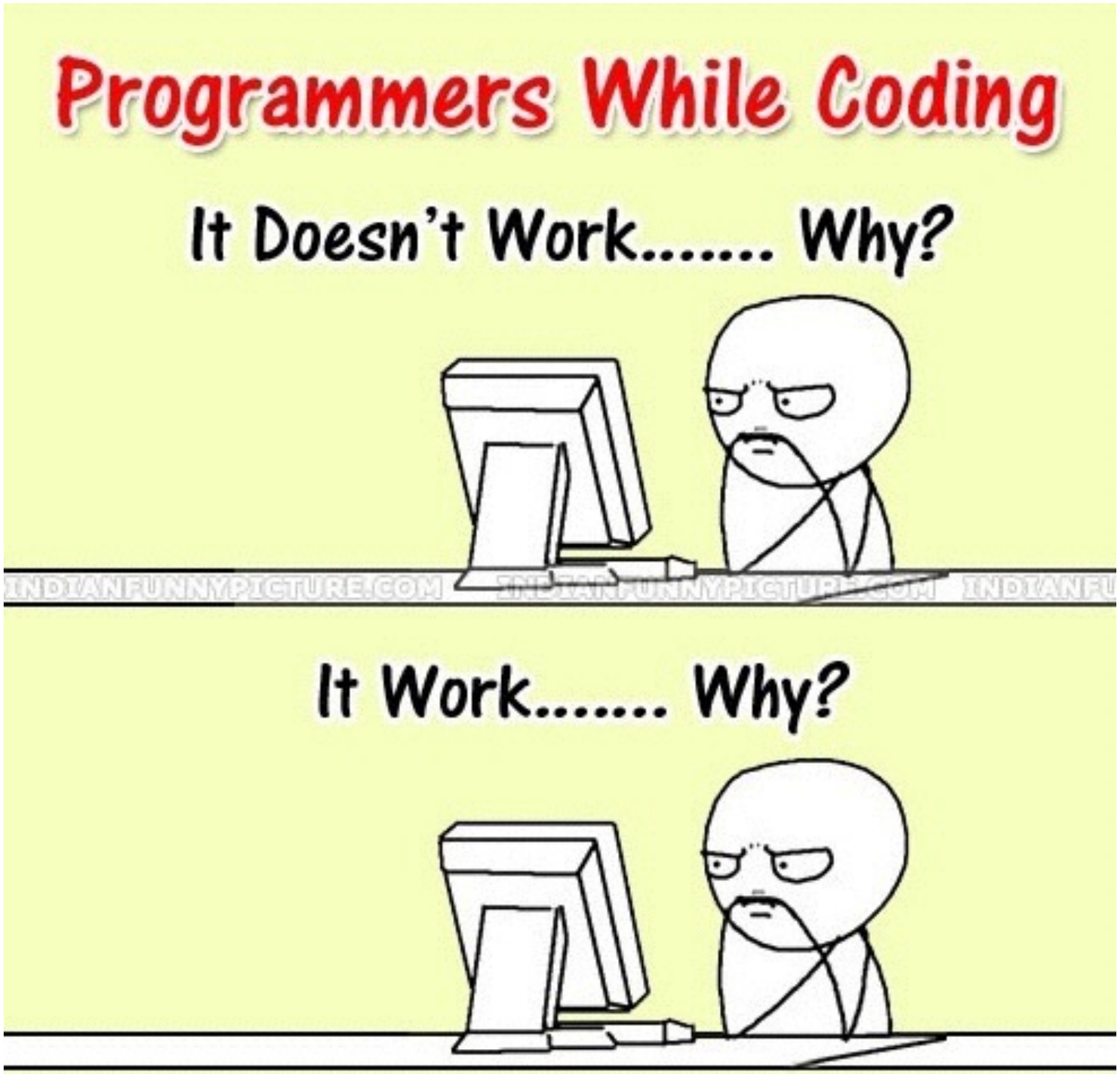
Наследование

```
01: class Author(User):
01:
02:     def __init__(self, name, age, posts=[]):
03:         super().__init__(name, age)
04:         self.posts = list(posts)
05:
06:     def publish(self, post):
07:         self.posts.append(post)
08:         print(post)
09:
10: admin = Author(”Jack”, 24)
```

Класс - предок

super() - получение доступа к родительским методам

Отладка (debug)



Отладка (debug)

```
01: while c >= 0:
02:     do_awesome_stuff()
```

Отладка (debug)

```
01: while c >= 0:
02:     do_awesome_stuff()
```

- обнаружение проблемы

Отладка (debug)

```
01: while c >= 0:
02:     do_awesome_stuff()
```

- обнаружение проблемы
- локализация проблемы

Отладка (debug)

```
01: while c >= 0:
02:     print(c)
03:     do_awesome_stuff()
```

- обнаружение проблемы
- локализация проблемы

Отладка (debug)

```
01: while c >= 0:
02:     print(c)
03:     do_awesome_stuff()
```

- обнаружение проблемы
- локализация проблемы (где сломалось?)
- воспроизведение (при каких условиях сломалось?)

Отладка (debug)

```
01: while c >= 0:
02:     print(c)
03:     do_awesome_stuff()
```

- обнаружение проблемы
- локализация проблемы (где сломалось?)
- воспроизведение (при каких условиях сломалось?)
- починка (почему это сломалось?)

Отладчики в python

Отладчики в python

- pdb – встроенный модуль для отладки

Отладчики в python

- pdb – встроенный модуль для отладки
- ipython и ipdb – “более лучшие” отладчики на основе pdb

Отладчики в python

- pdb – встроенный модуль для отладки
- ipython и ipdb – “более лучшие” отладчики на основе pdb

```
$ pip install ipython ipdb
```

Отладчики в python

- pdb – встроенный модуль для отладки
- ipython и ipdb – “более лучшие” отладчики на основе pdb

```
$ pip install ipython ipdb
$ python -m ipdb my_script.py
```

Отладчики в python

- pdb – встроенный модуль для отладки
- ipython и ipdb – “более лучшие” отладчики на основе pdb

```
01: import ipdb
02:
03: while c >= 0:
04:     print(c)
05:     do_awesome_stuff()
06:
07: def do_awesome_stuff():
08:     ipdb.set_trace()
09:     # остальной код …
```

Отладчики в python

- pdb – встроенный модуль для отладки
- ipython и ipdb – “более лучшие” отладчики на основе pdb
- pudb – графический консольный отладчик

Что важно знать?

Что важно знать?

- состояние системы на момент воспроизведения ошибки

Что важно знать?

- состояние системы на момент воспроизведения ошибки
- способ воспроизвести ошибку

Что важно знать?

- состояние системы на момент воспроизведения ошибки
- способ воспроизвести ошибку
- stack trace