textstatistics1.py

```
def get_char_frequencies(text):
    result = {}
    for char in text:
        if char in result:
            result[char] += 1
        else:
            result[char] = 1
    return result

def main():
    print(get_char_frequencies("abbbcc"))

if __name__ == '__main__':
    main()
```

```
C:\TEMP>python textstatistics1.py
{'a': 1, 'c': 2, 'b': 3}
```

textstatistics2.py

```
import collections

def get_item_frequencies(sequence):
    result = collections.defaultdict(int)
    for item in sequence:
        result[item] += 1
    return result

def get_char_frequencies(text):
    return get_item_frequencies(text)
```

```
C:\TEMP>python textstatistics2.py
defaultdict(<type 'int'>, {'a': 1, 'c': 2, 'b': 3})
```

textstatistics3.py

```
import re

def split_to_words(text):
    raw_word_list = re.split(r"\W", text, flags=re.UNICODE)
    # Get rid of empty items
    result = [word for word in raw_word_list if word]
    return result

def get_word_frequencies(text):
    return get_item_frequencies(split_to_words(text))

def main():
    print(get_char_frequencies("abbbcc"))
    print(get_word_frequencies("To be, or not to be"))
```

```
C:\TEMP>python textstatistics3.py
defaultdict(<type 'int'>, {'a': 1, 'c': 2, 'b': 3})
defaultdict(<type 'int'>, {'not': 1, 'To': 1, 'or': 1, 'to': 1, 'be': 2})
```

Классы и объекты

Определение класса:

```
class Class1:
    pass
```

Создание экземпляра класса:

```
object1 = Class1()
```

Инкапсуляция

```
class Class1:
    def _protected(self):
        print 'Это "защищенный" метод'

>>> object1 = Class1()
>>> object1._protected()
Это "защищенный" метод
```

Инкапсуляция

```
class Class1:
  def __private(self):
    print 'Это приватный метод'

>>> object1 = Class1()
>>> object1.__private()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: Class1 instance has
no attribute '__private'
```

Инкапсуляция

```
class Class1:
  def __private(self):
    print 'Это приватный метод'

>>> object1 = Class1()
>>> object1._Class1__private()
Это приватный метод
```

Полиморфизм

```
class Bird:
  def says(self):
    raise NotImplementedError

class Duck(Bird):
  def says(self): return u"Кря-кря"

class Cuckoo(Bird):
  def says(self): return u"Ку-ку"
```

Полиморфизм

```
>>> birds = [Duck(), Cuckoo()]
>>> for bird in birds:
...     print bird.says()
...
Кря-кря
Ку-ку
```

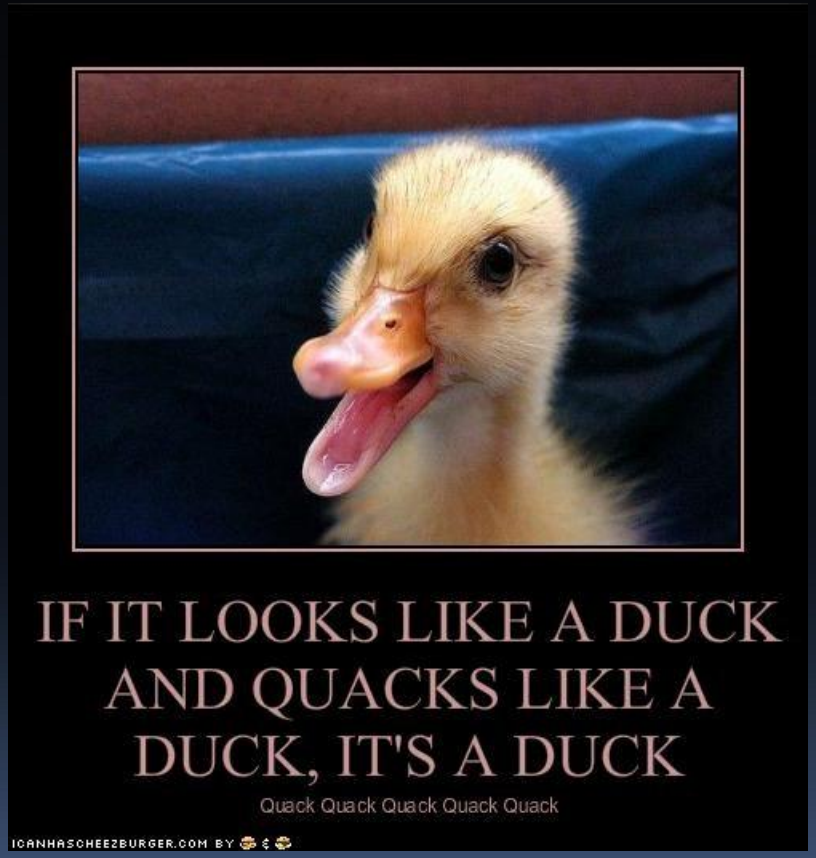
Полиморфизм

```
class Fox:
  def says(self): return u"???"

>>> animals = birds + [Fox()]
>>> for animal in animals:
...     print animal.says()
...
Кря-кря
Ку-ку
???
```

Утиная типизация

Если оно
выглядит как утка,
плавает как утка
и крякает как утка,
то это, наверное,
и есть утка



Утиная типизация