ООП: интеграция кода и данных

- Создание и применение типов, определяемых программистом для организации кода и данных
- Важная составляющая – интеграция типов данных с функциями, которые с этими данными работают
- Следующий наш шаг – переход от определения функций к методам, явно обозначающим такие отношения

Объектно-ориентированная программа

- Состоит из определений *классов* и *методов*
- Большинство вычислений реализуется в виде операций над объектами
- Классы часто моделируют вещи реального мира, а методы в классах соответствуют возможностям взаимодействия с ними
- *Методы* – это функции внутри класса, предназначенные как правило для работы с данными этого класса.
- 1-й параметр метода (*self*) обозначает представителя, посредством которого можно обращаться к методу. Передается в функцию неявно.
- Существует ряд специализированных методов, играющих особую роль для класса: __*init*__(), __*str*__(), __*add*__()…

Пример класса: время суток

```
# Определяет время суток. Атрибуты: hour, minute, second
class Time:
  # Выдача времени в стандартном формате
  def printTime(self):
    print('%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second))
```

- Использование класса и его метода

```
>>> start = Time()
>>> start.hour = 9
>>> start.minute = 45
>>> start.second = 0
>>> start.printTime()
09:45:00
>>>
```

Расширение класса Time

```
class Time:
  def printTime(self):
    print('%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second))
  def toInt(self):
    minutes = self.hour * 60 + self.minute
    seconds = minutes * 60 + self.second
    return seconds
  def toTime(self, seconds):
    minutes, self.second = divmod(seconds, 60)
    self.hour, self.minute = divmod(minutes, 60)
  def incTime(self, seconds):
    seconds += self.toInt()
    self.toTime(seconds)
```

Специализированный метод *init*()

- Имитирует *конструктор класса*, вызывается неявно при создании экземпляра
- Используется для инициализации атрибутов класса

```
class Time:
  def __init__(self, hour=0, minute=0, second=0):
    self.hour = hour
    self.minute = minute
    self.second = second
…
>>> time = Time()
>>> time.printTime()
00:00:00
>>> time1 = Time(9, 45)
>>> time1.printTime()
09:45:00
```

Специализированный метод *str*()

- Возвращает объект как строку, вызывается неявно в строковом контексте
- При его отсутствии работает стандартный метод для всех классов

```
>>> print(time1)
<Time object at 0x0000027518A8AA20>
class Time:
  def __init__(self, hour=0, minute=0, second=0):
    self.hour, self.minute, self.second = hour, minute, second
  def __str__(self):
    return '%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second)
>>> start1 = Time(9, 55)
>>> print(start1)
09:55:00
```

Перегрузка операторов

- Настройка поведения стандартных операций языка (*операторов*)
- В частности, метод __*add*__(), позволяет программировать операцию +

```
def intToTime(seconds):
  time = Time()
  minutes, time.second = divmod(seconds, 60)
  time.hour, time.minute = divmod(minutes, 60)
  return time
```

- Дополним класс *Time* методом __add__()

```
class Time:
  def __add__(self, other):
    seconds = self.toInt() + other.toInt()
    return intToTime(seconds)
```

Использование класса *Time*

```
>>> start = Time()
>>> start.hour = 9
>>> start.minute = 45
>>> start.second = 0
>>> start.printTime()
09:45:00
>>> print(start.toInt())
35100
>>> start.incTime(100)
>>> print(start.toInt())
35200
```

Перегрузка операторов - использование

```
>>> start = Time(9, 45)
>>> duration = Time(1, 35)
>>> print(start + duration)
11:20:00
```

- При применении операции + к объектам *Time* вызывается метод __*add*__()
- Когда печатается результат, Python вызывает метод __*str*__()
- Для каждой стандартной операции в Python предусмотрен соответствующий метод, подобно рассмотренному __*add*_ ().

Полиморфизм

- *Полиморфизм* – возможность использования единого механизма взаимодействия для различных элементов программы (типы, классы)
- В частности функция может оперировать объектами разных классов
- Полиморфизм упрощает повторное использование кода
- Функция *sum*() в Python вычисляет сумму членов различных составных числовых типов данных: *списков*, *кортежей* и *словарей*
- В ООП на Python конкретный объект, принадлежащий определенному классу, может быть использован таким же образом, как если бы он был объектом, принадлежащим другому классу или типу
- Класс *Time* предоставляет метод __*add*__(), поэтому функция *sum*() будет работать и с объектами этого класса

Полиморфизм – использование

```
>>> t1 = Time(7, 43)
>>> t2 = Time(7, 41)
>>> t3 = Time(7, 37)
>>> total = sum([t1, t2, t3], Time())
>>> print(total)
23:01:00
```

- В качестве второго ее параметра *sum*() указывается начальное значение суммы, в данном случае – нулевое время

Интерфейс и реализация

- Одна из задач ООП – облегчить сопровождение и обновление эксплуатируемого ПО
- Работоспособность программы при модификации ее компонент, при появлении и реализации новых требований
- Принцип проектирования, который помогает достигать подобных целей – *сокрытие информации*
- Состоит в отделении программных *интерфейсов* (описания методов) от их *реализаций*, сокрытых от внешнего взора
- Для объектов это также означает, что предоставляемые классом методы (способы обращения к ним) не должны зависеть от того, как представлены его атрибуты

Интерфейс и реализация – продолжение

- Класс *Time* представляет значение времени суток
- Его методы включают различную функциональность (__*str*__( ), __*add*__(), *printTime*(), *toInt*(), *toTime*(), *incTime*())
- Эти методы, как и многое другое в программировании, могут быть реализованы различными способами
- Детали реализации зависят от обстоятельств и принимаемых решений, в частности, от того, каким образом в классе представляется само время
- В качестве примера задействованы конкретные атрибуты класса *Time* – *hour*, *minute* и *second*
- Однако можно заменить их целым числом: количеством секунд, прошедших с полуночи, не меняя интерфейс класса
- Если интерфейс тщательно спроектирован и проработан, то его реализацию можно менять, сохраняя интерфейс, и другие зависящие части программы затронуты не будут

Словарь терминов

- Объектно-ориентированный язык. Язык с возможностями описания пользовательских типов и методов, поддерживающих концепцию ООП.
- ООП. Концепция программирования, в которой данные и операции над ними организованы в классы и методы.
- Метод. Функция, которая определена внутри класса и вызывается посредством экземпляров этого класса. Параметр *self* получает неявно.
- Имитатор конструктора. Метод __*init*__(), автоматически вызываемый при создании экземпляра класса.
- Перегрузка операторов. Настройка (изменение) поведения оператора таким образом, чтобы он работал с пользовательскими типами.
- Полиморфизм. Возможность использования общего интерфейса для различных элементов программы (таких как типы данных или классы).
- Сокрытие информации. Принцип, согласно которому предоставляемый классом интерфейс не должен зависеть от его реализации, в частности, от способа представления его атрибутов.