Методические указания

Урок 16.1. Написание модулей

Задачи урока:

- Написание модулей

0. Подготовка к уроку

До начала урока преподавателю необходимо:

1) Просмотреть, как ученики справились с домашним заданием
2) Прочитать методичку

1. Написание модулей

Учитель: Сегодня мы с вами поговорим о такой теме, как написание модулей. В процессе написания мы затрагивали тему работы, как со встроенными, так и сторонними модулями. Простейший модуль можно представить как обыкновенный файл python, в котором написаны различные функции, либо же переменные, такие, например, как число PI. Огромное преимущество модулей состоит в том, что написав единожды мы можем использовать его в дальнейшем просто импортируя. Таким образом, если со временем наша программа становится все больше в размерах, мы можем в дальнейшем разбить ее на отдельные модули.

Как же работать с модулями. Давайте для начала поработаем с уже написанными, а потом уже попробуем написать что то свое.

Для того, чтобы использовать какой либо модуль, сначала необходимо его импортировать(подключить). Подключить модуль можно с помощью команды import. Импорт модулей происходит в верхней части кода.

```
import random

print(random.randint(1, 10))
```

В данном примере мы импортировали модуль random, а потом воспользовались методом randint, который принимает два числа и выводит целое случайное число от первого до второго. Это не единственный метод данного модуля.

В Python можно указать псевдоним для метода или модуля, так как некоторые методы могут иметь достаточно длинное название. Для создания псевдонима используется команда as

```
import random as r

print(r.randint(1, 10))
```

В примере выше, мы указали, что для модуля random будет использоваться псевдоним r

Используя конструкцию import имя модуля мы по сути импортируем весь функционал данного модуля, также можно через запятую указать сразу несколько модулей.

При выполнении import происходят следующие операции.

1. Программа ищет исходный код модуля. Если его нет, выдается исключение ImportError.
2. Создается новый объект модуля. Он служит контейнером для всех глобальных определений в модуле. Иногда он называется пространством имен.
3. Исходный код модуля выполняется в только что созданном пространстве имен модуля.
4. Если выполнение проходит без ошибок, на стороне вызова создается имя, которое ссылается на новый объект модуля. Оно совпадает с именем модуля.

Для того, чтобы импортировать, например, только определенные определения необходимо воспользоваться инструкцией from *модуль* import *имя*

```
from random import randint

print(randint(1, 10))
```

В данном случае мы импортируем только метод randint и можем уже в коде обращаться к нему напрямую. Несмотря на то, что мы указываем в импорте только метод randint, при запуске нашего приложения у нас подгрузится весь код из модуля.
Также через запятую мы можем импортировать несколько методов

```
from random import randint, choice

print(randint(1, 10))
```

В данном случае также возможно использование псевдонимов

```
from random import randint as rint, choice as ch

print(rint(1, 10))
```

В модулях находятся не только методы, но и различные константы
Константа - переменная значение, которой мы не собираемся изменять и она постоянна. Например число PI

```
from math import pi

print(pi)
```

Math, тоже встроенный модуль(модуль стандартной библиотеки), в котором собраны различные математические константы и методы, например нахождения синуса, косинуса и т.п

Универсальный символ *, иногда используется для загрузки всего в модуле, кроме тех, методов, функций и переменных, чьи имена начинаются с подчеркивания:

```
from module import *
```

Модули могут точно управлять набором имен, импортируемых командой from *модуль* import *, для чего определяют список __all__:

Код модуля test.py

```
__all__ = ['func', 'MyClass']

number = 35       # Не экспортируется


def func():          # Экспортируется
  print('Hello')


class MyClass:   # Экспортируется
```

```
  pass
```

Код основного файла

```
from test import *


print(func())
obj = MyClass()
print(number)     # ошибка
```

Давайте предположим перед нами задача достать из списка случайный элемент. Как мы можем это сделать?
Один из вариантов выбрать случайное число от 0 до длины нашего списка - 1. И потом использовать данное число в качестве индекса

```
import random

a = [1, 2, 3, 4]

index = random.randint(0, len(a) - 1)
rand_num = a[index]
print(rand_num)
```

Но есть более простой вариант и он находится также в модуле random

```
import random

a = [1, 2, 3, 4]

print(random.choice(a))
```

Метод choice модуля random выбирает случайное значение из последовательности.

Учитель: Давайте теперь разберем вариант, когда два модуля импортируют друг друга. Предположим у нас есть 2 модуля

main

```
import test


def func_a():
```

```
 test.func_b()


class Base:
  pass
```

test

```
import main


def func_b():
  main.func_a()


class Child(main.Base):
  pass
```

В этом коде возникает странная зависимость порядка импортирования. Если сначала выполняется команда import main, все нормально, но если первым выполняется import test, происходит ошибка с сообщением о том, что значение main.Base не определено. Таким образом мы видим, что при команде import у нас выполняется весь код модуля и в данном случае у нас создается класс Base, от которого,в дальнейшем, мы можем наследоваться. Если же мы запускаем модуль main, то при импорте test, у нас происходит ошибка, так как мы пытаемся наследоваться от несуществующего класса.

Учитель: С подключением и использованием модулей думаю разобрались. Теперь давайте попробуем написать простой калькулятор и для него создадим модуль, котором опишем функции для расчета значений.

Создадим файл calc.py рядом с основным файлом и пропишем функции для сложения, вычитания, умножения, деления

```
def add(num1: float, num2: float) -> float:
   return num1 + num2

def sub(num1: float, num2: float) -> float:
   return num1 - num2

def mul(num1: float, num2: float) -> float:
   return num1 * num2

def div(num1: float, num2: float) -> float:
```

```
    return num1 / num2
```

Теперь напишем код в основном файле

```
from calc import add, sub, mul, div


class Calculator:
    def __init__(self) -> None:
        self.main()
    def main(self):
        print('Это калькулятор')
        while True:
            num1 = int(input('Введите первое число: '))
            num2 = int(input('Введите второе число: '))
            choice = int(input('Выберите необходимое действие 1: +, 2: -, 3: *, 4: /, 0: Выход\n'))
            match choice:
                case 0:
                    print('Для завершения нажмите Enter')
                    input()
                    break
                case 1:
                    print(add(num1, num2))
                case 2:
                    print(sub(num1, num2))
                case 3:
                    print(mul(num1, num2))
                case 4:
                    print(div(num1, num2))
                case _:
                    print('Неверный выбор')
obj = Calculator()
```

Вот и все. Мы создали свой простой модуль и использовали его в программе.
Новым и непонятным для вас в данном коде является операторы match и case. Это оператор выбора. Подобные конструкции есть во многих языках. Мы могли бы написать это и на обычных условиях. но раз существует данный оператор, то не помешало бы с ним и разобраться.

```
match значение которое проверяем:
    case необходимое значение:
        код выполняемый при совпадении
     case необходимое значение:
         код выполняемый при совпадении
```

Символ _ в последнем case обозначает для всего остального(аналог else в условиях)

3. Решение задач

Задача 1

Доработать в калькуляторе деление. Не забываем делить на 0 нельзя

```
def div(num1: float, num2: float) -> float:
    if num2 == 0:
        return 'Деление на 0'
    return num1 / num2
```

Дополнительно

Если на уроке остается время, то ученикам можно предложить начать прорешивать домашнее задание.

Домашняя работа

Задача 1