Рубежный контроль № 3: конспект по скриптовому языку

15 января 2024 г.

Дмитрий Лимонов, ИУ9-12Б

Скриптовый язык Python

1. Типизация и система типов языка.

Типизация в языке динамическая, неявная, сильная. Для определения типа данных можно использовать функцию `type()`. Вот примеры некоторых из них:

- **bool** - Неизменяемый тип данных. Булевы значения. Пример: True, False
- **NoneType** - Неизменяемый тип данных. Принимает единственное значение: None
- **int** - Неизменяемый тип данных. Целочисленные значения. Пример: 10, 20, 30
- **float** - Неизменяемый тип данных. Числа с плавающей точкой. Пример: 10.0, 20.9, 30.000003
- **complex** - Неизменямый тип данных. Комплексные числа. Пример: (3+4j)
- **str** - Неизменяемый тип данных. Тип данных для строк. Пример: “Hello, world!”, “1”, “[1, 2, 3]”
- **list** - Изменяемый тип данных. Массив, сохраняющий порядок, в котором элементы были добавлены, а так же допускает дубликаты. Пример: [10, 20, “thirty”, 40.2, True, None, None]
- **tuple** - Неизмянемый тип данных. Тип данных аналогичный типу list, отличается лишь неизменяемостью. Пример: (10, 20, “thirty”, 40.2, True, None, None)
- **range** - Неизменяемый тип данных. Итерируемый объект состоящий из цифр от начального значения и до конечного. Пример: range(1, 5), range(3, 7, 2), range(6, -5, -2)
- **set** - Изменяемый тип данных. Массив, не сохраняющий порядок добавления элементов, а также не хранящий дубликаты. Пример: {True, 40.2, 10, ‘thirty’, 20, None}
- **frozenset** - Неизменяемый тип данных. Тип данных аналогичный типу set, отличается лишь неизменяемостью. Пример: {True, 40.2, 10, ‘thirty’, 20, None}

- **dict** - Изменяемый тип данных. Ассоциативный массив, словарь. Пример: {“name”: “Дмитрий”, “age”: 18, “height”: 193.4}
- **bytes** - Неизменяемый тип данных. Тип данных, созданный для представления байтов. Пример: b’ ’

Это основные типы данных, которые широко применяются на практике. Также пользователь может создавать собственные классы, которые также будут определяться с помощью `type()`

2. Основные управляющие конструкции.

Python - язык высокого уровня, поэтому в нём представлены все основные управляющие конструкции:

Конструкция языка	Её значение
if [условие]: [тело] elif [условие]: [тело] else: [тело]	Ветвление
while [условие]: [тело]	Цикл с условием
for [переменная или _] in [итерируемый объект]: [тело]	Цикл с переменной
break	Прерывает выполнение любого из циклов выше
continue	Переходит на следующую итерацию любого из циклов
pass	Ничего не делает, “заглушка”
def [название и аргументы]: [тело]	Объявление функции
return [значение]	Возвращает значение из функции
try: [тело] except [любое исключение]: [тело]	Блок обработки исключений

3. Подмножество языка для функционального программирования

Способы обеспечить иммутабельность данных.

Как было отмечено в пункте 1, многие данные изначально неизменяемые (иммутабельные), а те объекты, которые всё-таки можно изменить, обладают иммутабельным аналогом (классу list, например, соответсвует класс tuple, и т.д.).

Если говорить не про встроенные классы, тогда существует декоратор, позволяющий “заморозить” класс:

```
from dataclasses import dataclass

@dataclass(frozen=True)
class Immutable:
    a: Any
    b: Any
```

Функции как объекты 1-го класса.

В языке программирования Python любая функция может быть присвоенна переменной, передана в качестве аргумента, возвращена другой функцией и в целом работа с ней идёт точно также как, например, с типом данных int, поэтому многие неочевидные конструкции не будут вызывать исключений, например:

```
def add(x, y):
    return x + y

def choosefunc():
    # some useful calculations
    return add

def interestingfunction(function):
    # more useful calculations
    return function(3, 5)

a = choosefunc()
print(a(1, 2))  # 3
print(interestingfunction(a))  # 8
```

Функции высших порядков. Встроенные функции высших порядков для работы с последовательностями.

Функции высших порядков - функции, которые в качестве аргумента принимают другие функции (прошлый пример, функция `interestingfunction(func)`) или возвращают их (функция `choosefunc()`). Раннее уже было сказано, что такие функции язык поддерживает, приведём примеры встроенных таких функций `map()` и `filter()`:

```
def multiply(x):
    return x * 10

a = [1, 2, 3, 4, 5]
b = list(map(multiply, a))  # [10 20 30 40 50]
# Также как в случае выше часто пишут lambda функции, например:
b = list(filter(lambda x: x % 2 == 0, a))  # [2 4]
# Декораторы также являются синтаксическим сахаром для функции высшего порядка,
# пример декоратора был приведён раннее (@dataclass(frozen=True))
```

4. Важнейшие функции для работы с потоками ввода/вывода, строками, регулярными выражениями

Поток ввода и вывода. Строки.

Для считывания с потока ввода можно применять два основных метода:

1. `input()` когда количество строк в введённых данных точно известно
2. Метод `.stdin` встроенной библиотеки `sys` в других случаях.

Для вывода же используют:

1. `print()` для вывода туда, откуда была запущена программа
2. `file.write()` или `print("some text", file=output)` для вывода в какой-либо файл, так например код снизу напишет одно и тоже в оба файла:

```
out1 = open("output.txt", mode="w")
out2 = open("output2.txt", mode="w")
# Важно, чтобы в обоих случаях mode был равен "w"
a = "Hello, world!\n"
out1.write(a)
print(a, file=out2, sep="")
# sep="" нужно, чтобы убрать лишний перенос строки
```

Python предоставляет множество различных методов, предоставляющих широкий функционал пользователю. Продемонстрировать работу каждой из них не получится, но все их можно получить с помощью простых команд:

```
print(dir(str))
# Выведет все методы без пояснения принципа их работы
help(str)
# Выведет все методы с пояснениями принципа их работы
```

Регулярные выражения.

Регулярные выражения - мощнейший инструмент для работы с какими-либо шаблонами. Python также предоставляет возможности по работе с ними во встроенной библиотеке `re`. Описывать принцип работы каждой из них также не имеет смысла, а получить справку по ним возможно также, как и в примере раннее:

```
import re

help(re)
print(dir(re))
```