Рубежный контроль № 3: конспект по скриптовому языку

25 декабря 2023 г.

Федуков Александр, ИУ9-12Б

Конспект по языку Python3

> Python - это высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью. Язык является полностью объектно-ориентированным в том плане, что всё в нем является объектами. Python является мультипарадигменным языком программирования, поддерживающим императивное, процедурное, структурное, объектно-ориентированное программирование, метапрограммирование, функциональное программирование и асинхронное программирование.

Типизация и система типов

Типизация в python строгая и динамическая.

Строгая, потому что язык не производит неявные преобразования типов и не получится сложить 1 + “1”

Динамическая, потому что типы объектов определяются в процессе исполнения программы и при создании перемнной ее тип можно не указывать.

Типы данных в python делятся на изменяемые и неизменеямые.

Неизменяемые: числа, строки, кортежи

Изменяемые: списки, словари, множества

Основные типы

Числа: `int float complex` *Числа, любой длины*

```
>>> a = 5
>>> print(a, "is of type", type(a))
5 is of type <class 'int'>
>>> a = 2.0
```

```
>>> print(a, "is of type", type(a))
2.0 is of type <class 'float'>
>>> a = 1+2j
>>> print(a, "is complex number?", isinstance(1+2j, complex))
(1+2j) is complex number? True
```

Списки: `list` *Упорядоченная последовательность элементов, не обязательно одного типа, обращение к элементу используют list_name[index]*

```
>>> a = [5,10,15,20,25,30,35,40]
>>> print("a[2] =", a[2])
a[2] = 15
>>> print("a[0:3] =", a[0:3])
a[0:3] = [5, 10, 15]
```

Кортежи: `tuple` *Как list, но неизменяемы*

```
>>> t = (5,'program', 1+3j)
>>> print("t[1] =", t[1])
t[1] = program
>>> t[0] = 10 # Приведёт к ошибке, т.к. кортежи неизменяемы
```

Строки: `str` *Набор символов, обращение к элементу с помощью str_name[index]*

```
>>> s = "Простая строка"
>>> s = '''многострочная строка'''
```

Множества: `set` *Неупорядоченная уникализированная последовательность*

```
>>> a = {5,2,3,1,4}
>>> print("a =", a)
a = {1, 2, 3, 4, 5}
>>> print(type(a))
<class 'set'>
```

Словари: `dict` *Хеш-таблицы или неупорядоченные наборы пар типа «ключ-значение». Работают, как ассоциативные массивы*

```
>>> d = {1:'value', 'key':2}
>>> print("d[1] =", d[1]);
d[1] = value
>>> print("d['key'] =", d['key']);
d['key'] = 2
>>> print("d[2] =", d[2]); # Приведёт к ошибке
```

Основные управляющие конструкции

Условия: `if elif else` *В случае something, выполняем соответсвующий do()*

```
if something1:
    do1()
```

```
elif something2:
    do2()
elif something3:
    do3()
# ...
elif somethingN:
    doN()
else:
    do_something_else()
```

Циклы: `for in range / while` *Повторяет expression пока не закончится range/истинно condition (break прерывает цикл, continue пропускает итерацию)*

```
for counter in range(start, stop, step):
    expression

while condition:
    loop_body
```

Исключения: `try except` *В случае ошибки в expressions to try выполняет expressions after error*

```
try:
    expressions to try
except error:
    expressions after error
else:
    expressions after no error
finally:
    expressions anyway
```

Таблица мутабельности различных типов

Иммутабельность данных можно обеспечивать переводом изменяемых типов к неизменяемым. Для list есть tuple, для set есть frozenset, dict также можно “заморозить” с помощью tuple

Класс	Описание	Мутабелен?
bool	булевый тип	-
int	целочисленный тип	-
float	число с плавающей запятой	-
list	список	+
tuple	кортеж	-
str	строка	-
set	множество	+
dict	словарь	+

Функциональное программирование

> Функции Python — это объекты первого класса. Их можно присваивать переменным, хранить в структурах данных, передавать в качестве аргументов другим функциям и даже возвращать в качестве значений из других функций.

В python два типа функций: именованные **def** и анонимные **lambda**

`def` *Принимает аргументы args и возвращает вычисленный result*

```
def func_name(args):
    func_body ...
    return result
```

`lambda` *Безымянная функция. По сути работает так же, как и обычная. Здесь x — это аргумент, а x*2 — это выражение, которое вычисляется и возвращается.*

```
double = lambda x: x*2
# Эквивалетно
def double(x):
        return x * 2
```

Функции высших порядков

> Функции, которые принимают другие функции в качестве аргументов и/или возвращают функции в качестве результатов, называются функциями высшего порядка или ФВП. Их можно использовать для инкапсуляции многократно используемого поведения и создания более абстрактного кода, о котором легче рассуждать.

Примерами ФВП являются: **filter()**, **map()**, **reduce()** , которые часто используются в паре с lambda функциями для обработки последовательностей.

`filter` *Принимает в качестве аргументов функцию и список и возвращает только те элементы этого списка, которые возвращают истину для этой функции*

```
# Пример: отбор четных чисел из списка
my_list = [1, 3, 4, 6, 10, 11, 15, 12, 14]
new_list = list(filter(lambda x: (x%2 == 0) , my_list))
print(new_list) # [4, 6, 10, 12, 14]
```

`map` *Принимает в качестве аргументов функцию и список и применяет эту функцию ко всем элементам списка, возвращая новую последовательность*

```
# Пример: удвоение всех элементов списка
current_list = [1, 3, 4, 6, 10, 11, 15, 12, 14]
new_list = list(map(lambda x: x*2 , current_list))
print(new_list) # [2, 6, 8, 12, 20, 22, 30, 24, 28]
```

`reduce` *Принимает в качестве аргументов функцию и список и производит свертку*

```
# Пример: сумма всех элементов
from functools import reduce
current_list = [5, 15, 20, 30, 50, 55, 75, 60, 70]
summa = reduce((lambda x, y: x + y), current_list)
print(summa) # 380
```

Некоторые важные функции

Работа с потоками ввода/вывода

Для вывода в стандартный поток вывода используют функцию **print()**.

Кроме того, после получения объекта потока *(будь то файл с помощью **open(file_path, open_mode)** или стандартные потоки ввода/вывода с помощью sys. sdtdin/sys.stdout)* с ним можно взаимодействовать при помощи методов **.read()** и **.write()**

```
# Пример: запись текста в файл
my_book = open("my_book.txt", 'w')
my_book.write('To be continued...\n')
my_book.close()
```

Работа со строками

```
# Конкатенация (сложение строк)
S1 + S2
# Повторение строки
S1 * 3
# Обращение по индексу
S[i]
# Извлечение среза
S[i:j:step]
# Длина строки
len(S)
# Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.find(str, [start],[end])
# Поиск подстроки в строке. Возвращает номер последнего вхождения или -1
S.rfind(str, [start],[end])
# Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
S.index(str, [start],[end])
# Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError
S.rindex(str, [start],[end])
# Замена шаблона
S.replace(шаблон, замена)
# Разбиение строки по разделителю
```

```
S.split(символ)
# Состоит ли строка из цифр
S.isdigit()
# Состоит ли строка из букв
S.isalpha()
# Преобразование строки к верхнему регистру
S.upper()
# Преобразование строки к нижнему регистру
S.lower()
# Начинается ли строка S с шаблона str
S.startswith(str)
# Заканчивается ли строка S шаблоном str
S.endswith(str)
# Сборка строки из списка с разделителем S
S.join(список)
# Символ в его код ASCII
ord(символ)
# Код ASCII в символ
chr(число)
```

Работа с регулярными выражениями

В Python для работы с регулярными выражениями есть специальный модуль, который использует стандартный синтаксис регулярных выражений.

`import re`

`re.match(pattern, string)` *Ищет pattern в начале строки string*

`re.search(pattern, string)` *Ищет только один pattern по всей строке string*

`re.findall(pattern, string)` *Ищет pattern по всей строке string*

`re.split(pattern, string, maxsplit=0)` *Разделяет строку string по подстрокам, соответствующим pattern*

`re.sub(pattern, repl, string)` *Заменяет в строке string все pattern на repl*

`re.compile(pattern)` *Собирает регулярное выражение в объект для будущего использования в других re-функциях*