**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана»**

**Факультет**: Информатика и системы управления

**Кафедра**: Теоретическая информатика и компьютерные технологии

**Рубежный контроль №3**

Конспект по языку **Python**

Выполнил

Студент группы ИУ9-11Б

Якубов Павел

**Конспект по скриптовому языку Python**

1. Типизация и система типов языка.

Python поддерживает динамическую типизацию, то есть тип переменной определяется только во время исполнения. Поэтому вместо «присваивания значения переменной» лучше говорить о «связывании значения с некоторым именем».

Добавить новый тип можно либо написав класс (class), либо определив новый тип в модуле расширения (например, написанном на языке C). Система классов поддерживает наследование (одиночное и множественное) и метапрограммирование. Возможно наследование от большинства встроенных типов и типов расширений.

Встроенные типы данных в Python

Питон работает с двумя категориями данных – встроенными типами (они поддерживаются по умолчанию) и специализированными (для операций с ними нужно подключение определенного модуля). К специализированным типам данных относятся, например, ***datetime*** (дата и время) и ***deque*** (двухсторонняя очередь).

Все встроенные типы данных в Python можно разделить на следующие группы:

**Числовые** – целые(***int***), вещественные(***float***), комплексные(***complex***) числа. Примечание: для максимально точных расчетов с десятичными числами в Python используют модуль decimal (тип данных Decimal), а для операций с рациональными числами (дробями) – модуль fractions (тип данных Fraction).

**Булевы** – логические значения ***True*** (истина) и ***False*** (ложь).

**Строковые** – последовательности символов в кодировке Unicode – ***str***.

**NoneType** – нейтральное пустое значение, аналогичное null в других языках программирования – ***None***.

**Последовательности** – списки(***list***), кортежи(***tuple***), диапазоны(***range***).

**Словари** – структура данных типа «ключ: значение» - ***dict***.

**Множества** – контейнеры, содержащие уникальные значения. Подразделяются на изменяемые(***set***) и неизменяемые(***frozenset***) множества.

**Байтовые** **типы** – ***bytes*** (байты), ***bytearray*** (изменяемая байтовая строка), ***memoryview*** (предоставление доступа к внутренним данным объекта).

Чтобы узнать тип данных, нужно воспользоваться встроенной функцией *type()*:

>>> a = 3.5

>>> type(a)

<class 'float'>

1. Основные управляющие конструкции

А) If/elif/else – ветка условного оператора – используется для проверки условий: если условие верно, выполняется блок выражений (называемый “if-блок”), иначе проверяется следующий блок “elif”. Если ни один из блоков с условием не был выполнен, выполняется другой блок выражений (называемый “else-блок”). Блоки “else” и “elif” являются необязательными.

**if** guess == number:

print('Поздравляю, вы угадали,') *# Здесь начинается новый блок*

print('(хотя и не выиграли никакого приза!)') *# Здесь заканчивается новый блок*

**elif** guess < number:

print('Нет, загаданное число немного больше этого.') *# Ещё один блок*

*# Внутри блока вы можете выполнять всё, что угодно ...*

**else**:

print('Нет, загаданное число немного меньше этого.')

Б) Оператор while – многократно выполняет блок операторов (обычно с отступом) до тех пор, пока проверка в заголовочной части оценивается как истинное значение.

**while** running:

guess = int(input('Введите целое число : '))

**if** guess == number:

print('Поздравляю, вы угадали.')

running = **False** *# это останавливает цикл while*

В) Цикл for – Оператор for..in также является оператором цикла, который осуществляет итерацию по последовательности объектов, т.е. проходит через каждый элемент в последовательности.

**for** х **in** ["spam", "eggs", "ham"]:

print(x, end=' ')

*#результатом этого цикла будет строка spam eggs ham*

Г) Операторы break/continue – прерывание выполнения цикла(while/for) изнутри цикла в случае ***break*** и продолжение выполнения цикла(while/for) и переход к следующей его итерации в случае ***continue***.

Д) try/except – конструкция, использующаяся для обработки исключений.

try:

a = int(input())

except:

print(«введено не число»)

Е) def func(arg1, arg2, …) – объявление функций внутри языка

**def** fibonacci(n):

**if** n **in** (1, 2):

**return** 1

**return** fibonacci(n - 1) + fibonacci(n - 2)

1. Подмножество языка для функционального программирования: способы обеспечить иммутабельность данных там, где это необходимо, функции как объекты 1-го класса, функции высших порядков, встроенные функции высших порядков для работы с последовательностями

А) Иммутабельность данных

К изменяемым типам данных в Python относят: list, dict, set и пользовательские классы. К неизменяемым: int, float, decimal, bool, string, tuple, range, frozenset. С помощью неизменяемых типов данных можно обеспечить иммутабельность данных. Но для ее обеспечения не стоит хранить изменяемые типы данных в неизменяемых – например, список внутри кортежа – хоть кортеж неизменяем, список внутри него легко меняется.

Б) Функции

Несмотря на то, что Python изначально не задумывался как язык функционального программирования, Python поддерживает программирование в стиле функционального программирования, в частности:

* **функция является объектом первого класса** (можно присвоить функцию переменной, и вызывать ее, используя имя этой переменной);
* **функции высших порядков** (функция может принимать другую функцию как аргумент);
* рекурсия.

Однако, в отличие от большинства языков, непосредственно ориентированных на функциональное программирование, Python не является чистым языком программирования и код не защищён от побочных эффектов

Встроенные функции высших порядков – map(), filter() – часто используются совместно с оператором lambda (безымянной функцией):

* Функция *map()* принимает два аргумента: функцию и аргумент составного типа данных, например, список. *map()* применяет к каждому элементу списка переданную функцию и возвращает <map object>, который можно затем переконвертировать в список – с помощью *list()*.
* Функция *filter()* принимает в качестве аргументов функцию и последовательность, которую необходимо отфильтровать(функция, передаваемая в *filter()* должна возвращать значение True / False) и возвращает <filter object>, который можно затем переконвертировать в список – с помощью *list()*.

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

**Функции ввода/вывода в консоль** (вывод может быть не только в консоль):

(выжимки из документации)

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys.stdout.

sep: string inserted between values, default a space.

end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

input(prompt=None, /)

Read a string from standard input. The trailing newline is stripped.

The prompt string, if given, is printed to standard output without a

trailing newline before reading input.

**Файловый ввод/вывод:**

Для открытия файла используется функция *open()*, которая возвращает файловый объект;

Для закрытия файла используется метод *close();*

Чтение данных из файла осуществляется с помощью методов *read(размер)* и *readline();*

Для записи данных файл используется метод *write(строка).*

**Функции и методы для работы со строками:**

Функции:

* str(n) — преобразование числового или другого типа к строке;
* len(s) — длина строки;
* chr(s) — получение символа по его коду ASCII;
* ord(s) — получение кода ASCII по символу.

Методы:

* find(s, start, end) — возвращает индекс первого вхождения подстроки в s или -1 при отсутствии. Поиск идет в границах от start до end;
* rfind(s, start, end) — аналогично, но возвращает индекс последнего вхождения;
* replace(s, new) — меняет последовательность символов s на новую подстроку new;
* split(x) — разбивает строку на подстроки при помощи выбранного разделителя x;
* join(x) — соединяет строки в одну при помощи выбранного разделителя x;
* strip(s) — убирает пробелы с обеих сторон;
* lstrip(s), rstrip(s) — убирает пробелы только слева или справа;
* lower() — перевод всех символов в нижний регистр;
* upper() — перевод всех символов в верхний регистр;
* capitalize() — перевод первой буквы в верхний регистр, остальных — в нижний.

**Функции для работы с регулярными выражениями** живут в модуле ***re***.

Основные функции:

| **Функция** | **Её смысл** |
| --- | --- |
| re.search(pattern, string) | Найти в строке string первую строчку, подходящую под шаблон pattern; |
| re.fullmatch(pattern, string) | Проверить, подходит ли строка string под шаблон pattern; |
| re.split(pattern, string, maxsplit=0) | Аналог str.split(), только разделение происходит по подстрокам, подходящим под шаблон pattern; |
| re.findall(pattern, string) | Найти в строке string все непересекающиеся шаблоны pattern; |
| re.finditer(pattern, string) | Итератор всем непересекающимся шаблонам pattern в строке string (выдаются match-объекты); |
| re.sub(pattern, repl, string, count=0) | Заменить в строке string все непересекающиеся шаблоны pattern на repl; |