1. **До рассвета**

0.1. Мы предполагаем, что вы имеете какой-то опыт программирования на каком-то языке. Поэтому мы не объясняем следующие понятия:

*программа*

*переменная*

*тип данных*

*оператор присваивания*

*условный оператор*

*оператор цикла*

*и, наверное, некоторые другие, про которые забыли написать.*

Если с этим есть проблемы, но вы хотите посещать наш курс, ~~позвоните 911~~ свяжитесь с Женей Баулиным или Михаилом Абрамовичем Ройтбергом. Или попросите своего вожатого найти кого-то из них.

0.2. То, что написано в следующих разделах – не учебник по языку Питон. Это только справочник, который поможет ориентироваться в том, что происходит на курсе. Не стоит искать в нем описания всех возможностей Питона и разбора сложных случаев.

0.3. То, что вы прочитали, называется *disclaimer.*

**Объявляется конкурс на лучший перевод этого слова (а) на русский и (б) на каталанский язык.**

Победители получат скидки! Удачи!

1. **Элементарные типы данных. Оператор присваивания.**

1.1. Основные элементарные типы данных в Python – ***числа*** и ***строки.*** Числа бывают двух типов: ***целые*** (в других языках: integer, int, цел) и вещественные (в других языках: float, reak, вещ).

**В Питоне нет описаний типов переменных!** Тип переменной определяется по оператору присваивания.

Пример 1.

**a, b = 324, 24**

**b = 24**

**pi = 3.14**

**m1 = “Ма”**

**m2 = 'ма'**

**print(a – b, ' ', pi, ' >> ', m1, m2)**

Здесь переменные **a** и **b** содержат целые значения; переменная **pi –** вещественное значение, а переменные **m1, m2** – строковое.

**При описании строки можно использовать**

**и двойные кавычки (“Ма”), и простые ('ма')**

Оператор в последней строке – это оператор печати.

Упражнение. Что будет напечатано, если запустить программу из примера 1? Запустите программу и проверьте себя.

**Оператор присваивания в Питоне: =**

**Чтобы проверить равенство значений величин используется знак: ==**

В Питоне **каждый оператор пишется в отдельной строке**

**2. Операции над элементарными типами.**

2.1. Арифметика целых чисел.

Сложение, вычитание и умножение записываются, как обычно:

2 + 3 = = 5

5 - 2 = = 3

3 \* 4 = = 12

Возведение в степень записывается так:

2 \*\* 3 = = 8

3\*\*2 == 9

И, наконец – деление

Деление: 15 / 3 = = 5.0

Остаток от деления: 19 % 5 = = 4

**В Python 3.x результатом деления целого числа на другое целое число будет вещественно число**.

Целочисленное деление записывается так:

7 // 2 = = 3.

1024 // 10 == 24

2.2. Арифметика вещественных чисел.

Никаких проблем: используем знаки +, -, \* и / обычным образом.

В выражениях с вещественными числами (или переменными) можно использовать и целые величины. Они автоматически переводятся в вещественные.

2.3. Строки.

Для строк есть две основные операции: слияние (обозначается +) и повторение (обозначается \*)

Пример 3.

**a = 'Фёдор'**

**b = 'Кондрашов'**

**c = '!'**

**print(a + ' ' + b + c\*3)**

Результат выполнения этой программы будет таким:

**Фёдор Кондрашов!!!**

Упражнение. Напишите программу, которая печатает

**Фёдор!!!!! Кондрашов!!!!!!!!**

**3. Списки**

3.1. Что такое список

В Питоне активно используются списки. Создать список можно следующим образом:

**list\_name = [value\_1, value\_2, …, value\_N]**

Элементами списка могут быть значения любого типа данных: целые числа, числа с плавающей запятой, строки и др. Даже список может быть элементом другого списка!

У каждого элемента списка есть свой номер, то есть адрес в этом списке, (часто вместо *номер* говорят *индекс*).

Пример 4.

**tea\_party = ['Rabbit', 'Bear', 'Fox', 'Turtle']**

**print(tea\_party[2])**

Результат выполнения этой программы:

**Fox**

***Вопрос*** Почему программа напечатала Fox, а не Bear ?

***Ответ:*** Потому, что в Питоне нумерация элементов списка начинается не с 0, а с 1.

Элементам списка можно присваивать значения так же, как обычным переменным:

**tea\_party[1] = 'Lion'**

**print(tea\_party)**

Результат:

**['Rabbit', 'Lion', 'Fox', 'Turtle']**

3.2. Списки равноотстоящих чисел (арифметические прогрессии)

Часто используются списки, элементами которых являются числа, идущие с определенным шагом, например,

а) {1, 2, …, 10};

б) {0, 0.1, …, 2.5};

в) {0, 2, 4, 6, …, 100}.

Для создания таких списков можно использовать специальную функцию range.

Вот как можно создать списки из примеров а) – б)

А = range(1, 10)

B = range(0, 2.5, 0.1)

C = range(0, 100, 2)

**Вызов функции range(N) создает список с элементами от 0 до N-1:**

3.3. Операции над списками.

Добавлять элементы в конец списка можно функцией append():

**tea\_party.append('Mouse')**

**print(tea\_party)**

Результат:

**['Rabbit', 'Lion', 'Fox', 'Turtle', 'Mouse']**

Чтобы получить часть списка нужно использовать конструкцию list\_name[a:b], которая выведет элементы списка с индексами от a до b-1. Например:

**tea\_party[1:3] = = ['Lion', 'Fox']**

Если индекс **a** не указан, то результат будет начинаться с элемента под индексом 0; если не указан индекс **b** – результат будет заканчиваться последним элементом списка. Например:

**tea\_party[:2] = = ['Rabbit', 'Lion']**

**tea\_party[3:] = = ['Turtle', 'Mouse']**

Также, можно использовать отрицательные индексы. Индекс -1 соответствует последнему элементу списка, -2 – предпоследнему и так далее.

3.4. Списки и строки.

3.4.1. Питон работает со строками и списками аналогичным образом. По сути, строка а Питоне рассматривается как список символов.

Перечисленные в п.2.3 операции над строками можно применять к спискам, а операции вырезки из п. 3.3 – к строкам.

Пример 5

**a = 'flimsy'**

**b = 'miserable'**

**c = b[0:1] + a[-4:]**

**print c**

Результат:

**mimsy**

??? Пример 6.

Склейка списков?

3.4.2. Если данные в строке разделены пробелами, или другими символами, удобно использовать функцию *split(char),* где char – символ-разделитель. Эта функция по строке строит список подстрок в соответствии с разделителями.

Пример.

**print('Beautiful is better than ugly'.split(' '))**

Результат:

**['Beautiful', 'is', 'better', 'than', 'ugly.']**

1. **Условия и циклы**
   1. Условный оператор

Условный оператор в Python выглядит следующим образом:

**a = 42**

**if a < 10:**

**print('the number is less than 10')**

**else:**

**print('the number is greater or equal to 10')**

**if a + b == 4:**

**print('printed when a + b equals four')**

**print('always printed')**

Замечание 1. Двоеточие после условия (в конце строки **if)** обязательно!

Замечание 2. После **if** и после **else**

Замечание 3. **В Питоне нет операторных скобок!**

Начало и конец группы операторов определяется по отступам.

**За отступами нужно ВНИМАТЕЛЬНО следить!**

* 1. Оператор цикла с проверкой условия.

Цикл while выглядит следующим образом:

**greetings = 1**

**while greetings <= 3:**

**print('Hello! ' \* greetings)**

**greetings = greetings + 1**

Эта программа напечатает следующее (команды в цикле выполнятся 3 раза):

**Hello!**

**Hello! Hello!**

**Hello! Hello! Hello!**

* 1. Оператор цикла с обработкой каждого элемента списка

Для выполнения определенного набора команд для каждого элемента списка можно использовать цикл for:

**NameList = ['Alice', 'Bob', 'Charley']**

**for name in NameList:**

**print('Hello, ' + name)**

Чтобы повторить команду ровно N раз, используйте функцию range(N), которая создает список с элементами от 0 до N-1:

**n = 10**

**for i in range(n):**

**print(i)**

Упражнение.

Что будет напечатано после выполнения такой программы;

**print (range(5, 12))**

Проверьте себя – выполните программу.

1. **Функции**

В Python можно создать свою функцию:

**def func(a,b):**

**return a+b**

Проверим, работает ли она:

**c = 3**

**d = 4**

**print(func(c,d))**

Результат:

**7**

1. **Модули и библиотеки**

В Python любой файл с программой называется ***модулем***. При написании программы можно использовать функции и переменные из сторонних модулей. Для этого сторонний модуль нужно вызвать. Это выполняется с помощью оператора *import:*

Пример.

**import math #** вызываем модуль math

**a = math.sin(math.pi) #** используем функцию sin() и переменную pi из модуля math

**print(a)**

Модули могут объединяться в группы, называемые библиотеками модулей или просто *библиотеками*. Чтобы вызвать модуль из библиотеки, используйте конструкцию *from* bib\_name i*mport* *module*\_name.

Пример.

**from matplotlib import pyplot** # вызываем модуль pyplot из библиотеки matplotlib

**pyplot.show()** # рисуем пустой график

Другой способ:

**import matplotlib.pyplot**

**matplotlib.pyplot.show()**

При импорте библиотек и модулей можно давать им свои имена («псевдонимы») и в программе использовать асевдонимы

**import numpy as np** # присоединяем модуль numpy и даем ему псевдоним np

**a = np.zeros((2,2))** # создаем матрицу 2 на 2, состоящую из нулей

# **zeros –** функцияиз модуля numpy

**print(a)**

**Д1. Словари**

Словарь, как и список, используется для хранения набора (множества) объектов. Отличие в том, что в списке в качестве индексов, т.е. адресов объектов, используются натуральные числа 0, 1, 2, …, а в словаре индексы явно задаются пользователем и могут быть любого типа. ??Могут ли индексы быть разных типов??

Пример.

**phones = {'Zoe':'232-43-58', 'Alice':'165-88-56'}**

**print(phones['Zoe'])**

Результат:

**232-43-58**

Первая строка создает словарь **phones.** в которомм хранится два элемента – строки **'232-43-58'** и **'165-88-56'.** Индекс первой из этих строк - **'Zoe';** индекс второй - **'Alice'**.

Изменение и добавление элементов происходит аналогично спискам:

**phones['Zoe'] = '658-99-55'**

**phones['Bill'] = '342-18-25'**

**print(phones)**

Результат:

**{'Bill': '342-18-25', 'Zoe': '658-99-55', 'Alice': '165-88-56'}**

**Словарь выделяется фигурными скобками, а список – квадратными**.

**Порядок элементов в словаре не регламентирован.**

Если в качестве индексов используются строки, нужно следить за регистром символов. Примерю

**d = {} #** Создаем пустой словарь

**d['key'] = 1**

**d['Key'] = 2**

**d['KEY'] = 3**

**print(d)**

Результат:

**{'KEY': 3, 'Key': 2, 'key': 1}**

Проверка, ***присутствует ли нужный индекс в словаре***, делается так:

**if 'Peter' in phones:**

**print("We know Peter's phone")**

**else:**

**print("We don't know Peter's phone")**

Результат:

**We don't know Peter's phone**

Чтобы ***удалить элемент*** из словаря, используйте оператор del:

**phones = {'Zoe':'232-43-58', 'Alice':'165-88-56'}**

**del phones['Zoe']**

**print(phones)**

Результат:

**{'Alice': '165-88-56'}**

**Д2. Работа с файлами**

Д2.1. Открываем файл.

Это делает функция open(file, mode), где file – имя файла, которое может включать путь, mode – режим открытия (‘r’ – для чтения, ‘w’ – для записи, при этом содержимое файла стирается, ‘a’ – для записи, при этом данные записываются в конец файла).

Пример:

**f\_idA= open('input.txt', 'r') # файл открывается на чтение**

**f\_idB = open('output.txt', 'w') # файл открывается на запись**

Здесь **f\_idA, f\_idB** – идентификаторы открытых файлов в программе.

При работе я файлом (чтении из файла, записи в файл, закрытии файла)

**используем идентификатор, а не имя файла в файловой системе.**

***В примерах считаем, что мы работаем с файлом с идентификатором f***

Д2.2. Читаем из файла.

Для получения данных из файла можно использовать *команду f.read().* Эта команда возвращает всё содержимое текстового файла f в виде одной строки.

При работе с файлом его **идентификатор указывается перед** именем функции: **f.**read().

Команда *f.readline()* возвращает одну строку из файла, причем каждая строка (кроме последней) содержит в конце символ “\n” – символ переноса строки.

Обратите внимание: Каждый вызов команды f.readline() возвращает последнюю непрочитанную строку файла.

Команда *f.readlines()* возвращает список, элементами которого являются все строки из файла.

Для считывания файла можно использовать и цикл for:

**for line in f:**

**print(line)**

Д2.3. Пишем в файл.

Функция записи в текстовый файл:

**f.write('Any data you want to write into file')**

Чтобы записать в файл данные, не являющиеся строкой, необходимо их конвертировать в строку – это делает функция *str(data):*

**School = ['ShMTB ', 2016, ' Barcelona ', 2, ' August - ', 19, ' August']**

**s = str(School)**

**f.write(s)**

Также, можно использовать цикл:

**for i in inscription:**

**f.write(str(i) + '\n')**

Данная программа запишет в файл каждый элемент списка в отдельную строку.

Д2.4. Закрываем файл.

После окончания работы с файлом его необходимо закрыть (не забывайте об этом!):

**f.close()**