**Введение**

Учебное пособие можно читать на сайте: https://ushliyork.github.io/project-site/

Страница проекта на github: https://github.com/UshliyOrk/python\_basics

Эта издание является учебным пособием для начинающих. Благодаря ему вы познакомитесь с языком программирования Python, а именно его третьей версией. Мы пройдём путь от простейших линейных алгоритмов до сложнейших конструкций с бесконечными циклами и фрактальным вызовом функций. Вы можете обучаться с помощью этого издания дома или на дополнительных занятиях в школе, при наличии компьютера. Для программирования вам понадобиться прежде всего установить сам Python с официального сайта, код можно писать в любом редакторе текста, но удобнее всего это делать в специализированном редакторе, таком, как например PyCharm. Чтобы скачать PyCharm, перейдите по ссылке и скачайте любую версию:

https://www.jetbrains.com/ru-ru/pycharm/download/#section=windows

Все примеры нужно запускать самому для полного понимания. Их можно скачать и запустить: <https://github.com/UshliyOrk/python_basics>

Сокращения:

Яп – язык программирования

**Глава I. Основы работы яп Python. Переменные, их типы,**

**и операции над ними**

**Параграф 1. Основы**

Все яп переводятся в бинарный код для обработки процессором, этим занимается компилятор. Но “компилятор” в Python называется интерпретатор, он переводит python код в байт код. Это позволяет сильно упростить язык. Но не думайте, что у Python нет минусов, он работает заметно медленнее других яп.

В Python существуют различные библиотеки, расширяющие возможности программирования. Чтобы ими пользоваться нужно прописать команду import.

Пример 1

import math  
import math as mt  
from math import \* #импортирует все из модуля math   
from math import sqrt  
  
  
# однострочный комментарий  
"""  
многострочный комментарий  
"""  
math.sqrt(144)  
mt.sqrt(144)  
sqrt(144)  
sin(45)

Чтобы вывести информацию в консоль, в Python используется команда print(). Если вы хотите вывести число, просто напишите его в скобках, если же вы хотите вывести строку, то она должна быть заключена в любые кавычки, чтобы вывести переменную, нужно просто ввести её название, если нужно вывести несколько значений подряд, они записываются через запятую.

Пример 2

print("Hello world!", 10)

Текст можно выводить форматировано, используя обратный слеш и некоторые буквы английского алфавита.

\\ - так можно вывести \

\’ или \” – выводит кавычки

\n – выводит следующий далее текст с новой строки

\t – добавит табуляцию

Пример 3

print('Hello\nworld!')

Python значения, написанные через “,”, выводит через пробел, пробел можно заменить чем угодно, с помощью sep.

Пример 4

print('Hello,','Brad' , 'and Vlad', sep='=')

Также у функции print есть параметр end, по умолчанию он равен “\n”.

Пример 5

print('Hello,','Bard' , 'and Vad', end='novaya stroka')

**Параграф 2. Переменные**

Переменные – это основа любого языка программирования. У переменных всегда есть названия. В Python название переменной не может начинаться с числа, также вы, конечно, можете назвать переменную кириллицей, а кто-то даже сможет мефодицей, но так не принято делать. Переменные хранят в себе значения, и, как ни странно, их можно менять. Также во многих языках программирования присутствуют константы. Константы – это как переменные, но их нельзя менять. В Python нет констант, но вместо этого вы можете написать название переменной большими буквами, как в настоящем языке программирования. Оператором присвоения является знак “=”.

Переменные бывают нескольких видов:

* Целочисленные – int (от англ. integer)
* Числа с плавающей точкой – float
* Строковые – str (от англ. string)
* Смысловые – bool
* Список – list
* Словарь – dict

**Параграф 3. Операции над целочисленными переменными и переменными с плавающей точкой**

Помимо основных вычислительных операций присутствуют такие как целочисленное деление “//” и остаток от деления “%”.

Пример 6

#Основные мат. операции  
print('x+y=', x+y)  
print('x-y=', x-y)  
print('x\*y=', x\*y)  
print('x/y=', x/y)  
print('x%y=', x%y)#наход. остатка при /  
print('x//y', x//y)#наход целой части при /  
print('x\*\*y', x\*\*y)#возвести в степень

**Параграф 4. Операции над строками**

Строки можно складывать и умножать. Если вывести сложение строк, они будут написаны слитно.

Пример 7

a = "abc"  
b = 'xyz'  
c = a + b  
print("a =", a)  
print("b =", b)  
print("(a + b) =", a + b)  
print(c \* 3)

К строкам можно обращаться по индексу, индекс первого символа строки – 0, индекс последнего символа строки – -1. Также с помощью индексов можно извлекать срезы.

Пример 8

s = "Hello, beautiful world!"  
print(s[5], s[-3])  
print(s[3:5]) # срез  
print(s[:]) # полный срез  
print(s[3:1:5]) # срез с шагом

Если нужно узнать количество символов в строке, используется функция len().

Пример 9

Stroka = "1234"  
print(len(Stroka))

У каждого символа есть свой код в кодировке ASCII (American standard code for information interchange). Его можно узнать с помощью команды ord. Также из числа можно получить букву с помощью команды chr.

Пример 10

S = "Poulic"  
P = ord(S[0])  
print("P =", P)  
print(chr(P))

**Методы строк**

# 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()

# Состоит ли строка из цифр или букв

S.isalnum()

# Состоит ли строка из символов в нижнем регистре

S.islower()

# Состоит ли строка из символов в верхнем регистре

S.isupper()

# Состоит ли строка из неотображаемых символов (пробел, символ перевода страницы ('\f'), "новая строка" ('\n'), "перевод каретки" ('\r'), "горизонтальная табуляция" ('\t') и "вертикальная табуляция" ('\v'))

S.isspace()

# Начинаются ли слова в строке с заглавной буквы

S.istitle()

# 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)

# Переводит первый символ строки в верхний регистр, а все остальные в нижний

S.capitalize()

# Возвращает отцентрованную строку, по краям которой стоит символ fill (пробел по умолчанию)

S.center(width, [fill])

# Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию)

S.count(str, [start],[end])

# Возвращает копию строки, в которой все символы табуляции заменяются одним или несколькими пробелами, в зависимости от текущего столбца. Если TabSize не указан, размер табуляции полагается равным 8 пробелам

S.expandtabs([tabsize])

# Удаление пробельных символов в начале строки

S.lstrip([chars])

# Удаление пробельных символов в конце строки

S.rstrip([chars])

# Удаление пробельных символов в начале и в конце строки

S.strip([chars])

# Возвращает кортеж, содержащий часть перед первым шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий саму строку, а затем две пустых строки

S.partition(шаблон)

# Возвращает кортеж, содержащий часть перед последним шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий две пустых строки, а затем саму строку

S.rpartition(sep)

# Переводит символы нижнего регистра в верхний, а верхнего – в нижний

S.swapcase()

# Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний

S.title()

# Делает длину строки не меньшей width, по необходимости заполняя первые символы нулями

S.zfill(width)

# Делает длину строки не меньшей width, по необходимости заполняя последние символы символом fillchar

S.ljust(width, fillchar=" ")

# Делает длину строки не меньшей width, по необходимости заполняя первые символы символом fillchar

S.rjust(width, fillchar=" ")

Для полного усвоения рекомендуется по пробовать на деле все эти методы и вам наверно кажется, что большая их часть не пригодится, но это не так.

**Параграф 5. Списки и операции над ними**

Список – последовательность данных разных типов. У списка тоже есть нумерация, такая же как у строки. Команда len также справедлива для списков. Можно делать срезы списков. Метод index принимает значение и возвращает индекс этого значения. Метод count принимает значение, а возвращает количество его повторений. Метод sort сортирует список. Метод append принимает значение добавляет его в конец списка. Метод remove удаляет полученное значение из списка. Метод pop удаляет значение по индексу. Метод extend склеивает два списка, тоже самое что и сложение двух списков. Метод insert принимает индекс и вставляет значение. Также существуют функции: min, max, sum. Можно проверять, есть ли значение в списке с помощью in или not in.

Пример 11

l1 = [1, 2, 4, 3, 5, 4, 3, 6, 1, 2, 8, 7, 9]  
l2 = ['Ann', 'Paul', 'Rodion', 'Max', 'Matt', 'Billy', 'Thomas']  
print(l2.index('Paul'))  
print(l1.index(2, 4)) # ищем индекс двойки, но начиная с четвертого индекса  
print(l1.count(3))  
l1.sort()  
print(l1)  
l2.sort(reverse=True) # в обратном порядке  
l2.append('Tom')  
print(l2)  
l1.remove(1)  
print(l1)  
name0 = l2.pop(0)  
print(l2)  
print(name0)  
l1.extend([1, 2])  
print(l1)  
l1.insert(0, l2) # мы получили вложенный список  
print(l1)  
print(l1[0][3]) # таким образом я обратился к вложенному списку и выведу то что находится в нем под индексом 3

**Параграф 6. Словари**

Словари – это не упорядоченные списки. Словарь содержит пары ключ : значение. К элементам словаря нужно обращаться по ключу. Словари могут пригодиться при де кодировке или кодировке. В целом словари самый мало используемый тип, но упомянуть их стоит. Запись словарей выглядит так.

Пример 12

dct1 = {'P': 'Paul', 'A': 'Ann', 'R': 'Rodion'}

словарь можно создать и так.

Пример 13

l1 = ['P', 'A', 'R']  
l2 = ['Paul', 'Ann', 'Rodion']  
dct2 = dict.fromkeys(l1, l2) # пример случая , когда хотел одно, а получилось другое  
print(dct2)  
dct2['P'] = 'Paul'  
dct2['R'] = 'Rodion'  
dct2['A'] = 'Ann'  
print(dct2) # теперь все так как и хотел

**Параграф 7. Переменные логического типа**

Переменные логического типа хранят в себе одно из двух значений True или False.

На этом все про логические переменные.

**Итоги главы**

В этой главе мы познакомились со всеми переменными и с тем, что с ними можно делать,

с основами работы яп python.

Задания по этой главе есть в конце пособия, настоятельно рекомендуем их просмотреть.

**Глава II. Алгоритмические конструкции. Логические операции**

**Параграф 1. Виды алгоритмических конструкций**

Существует 3 типа алгоритмических конструкций: следование, ветвление, цикл.

Алгоритмическая конструкция «следование» представляет собой прямое выполнение последовательно записанных команд. Во всех выше приведённых примерах использовалась конструкция «следование».

Ветвление – это конструкция, имеющая условие, которое может быть только верным или неверным, и от этого зависит последовательность команд.

Цикл – это конструкция, имеющая условие, пока это условие является верным, все команды в теле цикла повторяются.

Алгоритмические конструкции тоже одна из важнейших частей яп, наря ду с переменными.

**Параграф 2. Ветвление**

Перед тем как приступить к конструкции «ветвление», стоит оговорить, что если вам нужно занести в переменную её же изменённое значение, то можно использовать такой синтаксис:

Пример 14

a += 2 # + можно заменить на любой знак, а в данном случае становиться 3

Условием конструкции может быть сравнение, для этого используют знаки <, >, ==, != - не равно.

Также условием может быть любая команда, возвращающая значение True/False.

Пример 15

a = '1'  
if a.isdigit():  
 print("Hello world!")

Пример 16

a = 1  
if a != 100:  
 print("Hello world!")  
 a += 2   
elif a == 3:  
 a += 99  
 print(a)  
else:  
 pass

Как понятно из примеров, ветвление начинается со слова if, затем следует условие, а потом с новой строки, с отступом в одну табуляцию (Tab\*1/Space\*4), команды, которые нужно выполнить, если условие окажется верным. Если нужно продлить конструкцию на ещё одно условие используется elif. Else (англ. “иначе”) обычно закрывает всю конструкцию. Если вам очень хочется закончить конструкцию на else, вы можете использовать pass. В конструкции не обязательно должны присутствовать else и elif. Ветвление работает таким образом, что если он

находит первое верное условие, все остальные команды, входящие в конструкцию, не выполняются. Можно также передать несколько условий

**Параграф 3. Циклы**

В Python есть два варианта циклов: с условием и циклы, повторяющиеся определённое кол-во раз.

Цикл while имеет условие и выполняется, пока условие верно.

Цикл while определяется так:

While <условие>:

<тело цикла>

Пример 17

a = 0  
while True:  
 print("Hello")  
 a += 1  
 if a == 100:  
 break

Break завершает цикл while.

Цикл for присваивает переменной поочерёдно все значения переданного списка

Команда range создаёт список

Пример 18

list1 = []  
for i in range(10): # конечное число 9  
 list1.append("Python " + str(i))  
  
  
list2 = []  
for t in range(3,6): # от 3 до 5  
 list2.append("World " + str(t))  
  
  
for s in range(2, 123, 4): # от 2 до 122 с шагом 4  
 print(str(s) +" =", float(s))  
  
  
for r in list1: # r поочерёдно принимает все значения list1  
 print(r)

break завершает и цикл for.

После цикла fore может стоять блок else, он выполнится только если цикл успешно закончился

for i in range(2, 5):  
 print(i)  
else:  
 print('цикл успешнозавершен')  
  
for i in range(2, 5):  
 print(i)  
 break  
else:  
 print('цикл успешнозавершен')  
print('цикл прерван')

логические выражения можно объединить, используя конъюнкцию, дизъюнкцию и отрицание

and – конъюнкция

or – дизъюнкция

not – отрицание

a = True  
b = False  
x = 0  
while a and not b:  
 x += 1  
 if x == 3:  
 a = False

**Итоги главы**

В этой главе мы узнали, что такое алгоритмические конструкции, их виды, условные операции и как они работают.

**Глава III**

**Объектно-ориентированное программирование(ООП)**

Объе́ктно-ориенти́рованное программи́рование (ООП) — [методология программирования](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), основанная на представлении программы в виде совокупности [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), каждый из которых является экземпляром определённого [класса](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), а классы образуют иерархию наследования

Проще говоря нужно разбить сложный процесс на множество подпроцессов для упрощения понимания и ускорения работы кода.

**Параграф 1. Функции**

Функции позволяют нам выполнять часть кода множественное количество раз, в функции можно передавать разные значения и получать ответ от нее.

Функция определяется так:

def <название функции>(<параметры>):

<тело функции>

Пример 19

# создание функции  
def hello():  
 print('Hello, World!')  
# вызов функции  
hello()

# функция с параметром  
def user\_hello(name):  
 print('hello, ', name)  
user\_hello('Paul')

# можно передать несколько параметров  
def summ(stop, s, u, x):  
 for i in range(stop):  
 s += u  
 u += x  
 # функция возвращает переменную  
 return s  
# присваиваем переменной то, что вернула функция  
s = summ(23, 10, 14, 4)  
print(s)

После return выполнение функции прерывается.

Используя global можно назначить общие для всех функций переменные

Функции позволяют оптимизировать код и не писать одно и тоже множество раз.

usr = []  
def user(name, age):  
 global usr  
 usr.append(name)  
 usr.append(age)  
  
def print\_user():  
 print(usr)  
user('Rodion', '15')  
print\_user()

**Параграф 2. Точка входа**

Как вы уже знаете в Python можно импортировать другие библиотеки, но также вы можете импортировать в свой файл другой ваш файл, при этом он исполнится. Но если вам этого не нужно придется воспользоваться точкой входа.

Python интерпретатор автоматически создает переменную с названием \_\_name\_\_ и значением \_\_main\_\_, но только если вы запускаете сам файл, а не импортируете его

Пример 20.1

def square():  
 print(" -------")  
 print("| |")  
 print("| |")  
 print(" -------")  
square()

пример 20.3

import пример201  
for i in range(1, 4):  
 пример201.square()  
# рисует на один квадрат больше чем хотелось бы

пример 20.2

def square():  
 print(" -------")  
 print("| |")  
 print("| |")  
 print(" -------")  
  
# знакомтесь, точка входа  
if \_\_name\_\_ == "\_\_main\_\_":  
 square()

пример 20.4

import пример202  
for i in range(1, 4):  
 пример202.square()  
# рисует столько сколько нужно

**параграф 3. Классы**

Класс нужен для описания чего-то большого и хранения множества функций и переменных, класс может описывать персонажа игры или поле битвы, сложную математическую формулу и вообще все что душе угодно.

Класс задается так:

Classs <имя> (параметры):

<тело класса>

Класс похож на функцию, но это только на первый взгляд. Есть не мало отличий.

В классе нельзя просто создать переменную или вызвать функцию, нужно сказать, что она принадлежит ему через self

В классах все записывается в функциях. Функция с названием \_\_init\_\_ будет запускаться при каждом создании экземпляра класса

Пример 21

class user1():  
 def \_\_init\_\_(self):  
 self.NA = []  
 name = input('введите имя первого пользователя --> ')  
 age = int(input("введите его возраст --> "))  
 self.age\_name(name, age)  
 def print\_all(self):  
 print(self.NA)  
  
 def age\_name(self, name, age):  
 self.NA.append(name)  
 self.NA.append(age)  
  
if \_\_name\_\_ =="\_\_main\_\_":  
 # создание экземпляра класса  
 usr = user1()  
 # использование функции класса  
 usr.print\_all()

**Итоги главы**

Мы немного разобрались в ООП. Этих знаний достаточно для проектов средней сложности, а углубляться можно до бесконечности и конечно в одной главе всего не объять.

Немного отходя в сторону от основной темы, существует библиотека Qt и программа Qt Designer,

С помощью которых можно легко, например, написать калькулятор с графикой.

Через библиотеку pyinstaller можно осуществить компиляцию кода в исполняемый файл.

Обо всем этом можно узнать из дополнительных источников в конце

Мы лишь приложим в дополнительных материалах пример и инструкцию по установке библиотек через pip.

**Заключение**

Вот вы и узнали основы яп Python, но до настоящих программистов ещё далеко, нужно больше практики и углубления в язык. Вам необходимо попробовать создать какой-нибудь большой проект с использованием всех полученных знаний. Мы надеемся, что наше пособие помогло вам и теперь вы готовы к программированию.