Основы программирования (КИСиП)

Контрольная работа 21/22 года обучения

К контрольной работе приступать после изучения: **лекции** , **литература** и **Глоссарий (основные определения)**.

Проходной балл 60% (максимально 40% за тестовые вопросы и 60% за программу). Программу проверяют в течении 3 дней.

Программы в Python

Любая программа состоит из множество строк кода и различных логических конструкций.

```
print('hello')
```

является простой программой которая выведет нам слово hello на экран.

Если вы будете использовать PyCharm то в нем можно напрямую вызывать код на исполнение и он запустится с определенной точки. Про точки вызова мы поговорим на одном из следующих занятий и вам про это расскажет на лекции Владимир Владимирович.

В данном случае точкой входа будет являться первая строчка файла который мы вызовем.

В других различных IDE существуют свои варианты запуска и отладки ваших программ. Более подробно описано в документации к ним, которую вы можете найти на сайте разработчиков. Большинство из них уже переведены на простой и понятный русский язык.

Так же для Python существует режим интерпретатора, в котором мы можем выполнять действия напрямую. (`1+1, 'hello'`).

В случае если вы будете использовать VSC удобно будет использовать командную строку для вызова программ. В случае если вы установите точки останова программы для отладки, при вызове программы из консоли VSC точки остановки будет учитывать.

Запуск программы выполняется с помощью вызова основного файла, где у вас находится точка входа в ваше приложение. Хорошим тоном будет выделять ее явно в виде блока main.

Исполняемые файлы Python

Все файлы с кодом программ на Python будут иметь расширение .py

Простая просьба, которая сэкономит вам и мне много времени, создавайте файлы с расширение .py через VSC, PyCharm и прочие IDE. Тогда у файла изначально будет кодировка UTF-8.

Кодировка UTF-8

```
# -*- coding: utf-8 -*-
```

В данный момент это самый распространенный стандарт кодирования символов для хранения и передачи символов Юникода. Про кодировки вы более подробно узнаете на курсе информатики. Данная строчка в начале файлов с расширение .py, обозначает всем, кто будет использовать и модифицировать нашу программу, что файл должен распознаваться именно в ней.

Синтаксис языка Python

Соглашения

В данный момент следует придерживаться только одного условия:

Вся строка кода должна вмещаться в один экран. В случае если она будет длинее ставите символ экранирования \ и пишете на следующей строке. Так можно понять, что выполняет данный блок кода не пролистывая экран вправо-влево.

Со временем у нас появятся еще некоторые правила, но я их озвучу заранее и они будут указаны в репозитории.

Комментарии

```
# простой комментарий с текстом
print("Hello World")  # Встроеный комментарий с текстом
```

С помощью комментариев вы можете описывать что выполняет определенный кусок кода. В комментариях возможно указывать, любую информацию. При выполнении кода интерпретатор просто пропустит этот участок.

Так же стоит затронуть многострочные комментарии.

```
""" Блок
с
многострочным
комментарием """
```

Такие комментарии чаще используются для документирования кода. Они уже несут смысловую нагрузку в том, что их можно автоматически собирать по всему документу и формировать один файл или базу знаний для описания тех или иных функций кода.

Первые несколько занятий вы будете размещать в блоках многострочных комментариев текст вашей задачи. Поскольку большинство ваших первых решений будут хранится лишь в одном файле — это будет просто и практично.

Значение pass

Оператор pass является оператором заполнения. Он позволяет программе невыполнять никаких действий - буквально "сделать ничего". Обычно это требуется когда наличие какого-либо кода необходимо синтаксически, но не требуется по логике работы программы.

К примеру когда программа выполняет определенные действия, но при достижении какого-то определенного значения, она должна пропустить его и продолжить выполнять обычную работу.

Отступы

Для ограничения конкретного блока кода в языке Python используются отступы. По единому стандарту используется 4 пробела. Отделять блоки кода друг от друга нам потребуется для логического разделения программы.

Типы данных в Python

Строки

Для хранения внутри программы каких-либо текстовых значений используются строки.

Такие данные обозначаются с помощью одинарных, двойных или тройных кавычек. Тип кавычек для обозначения строки вы можете выбирать по своему вкусу, но следует помнить, что кавычки являются экранируемыми. Соответственно тип кавычек которые начинает строку, должен ее и завершать.

```
''' Блок кода в котором можно БЕЗОПАСНО использовать 'одинарные' и "двойные" кавычки '''
```

Благодаря такому правилу в строках можно хранить апострофы и цитаты.

Числа

Для хранения каких-либо данных требуются не только строки, но и числа. Конечно в строках можно хранить цифры, но эти цифры будут для программы нести такой же смысл как и буква "а" или тире. Чтобы цифры обрели смысл и стали числами, требуется хранить их отдельно от строк и не смешивать.

Как и в жизни числа в языке Python бывают целочисленные и вещественные.

В Python 3 целые числа поддерживают все возможные математические операции.

Математические операции

Обозначение	Операция
+	Сложение
x - y	Вычитание
x * y	Умножение
x / y	Деление
x // y	Получение целой части от деления
x % y	Остаток от деления
-x	Смена знака числа
abs(x)	Модуль числа
x ** y	Возведение в степень

Так же в Python 3 реализована поддержка длинной арифметики.

Отличия вещественных от целых заключаются в способе хранения их в памяти компьютера и количеством занимаемой памяти на одно число. Но вещественные числа следует использовать только в тех случаях когда точность не имеет критического значения.

Так же вещественные числа можно преобразовывать в целые числа с помощью функции `int()`. При преобразовании число округляется в меньшую сторону. Функция `int()` может также преобразовывать и строки если в них хранится только число. Для преобразования строки в вещественное число следует использовать функцию `float()`.

О функциях мы поговорим чуть позже.

Переменные

Любые значения в программе будь то числа, строки или любая информация, хранятся с помощью некоторых идентификаторов. Эти идентификаторы принято называть переменными.

Для более простого представления можно представить коробку в которую вы кладете информацию написанную на листочке. Каждый раз когда вам требуется эта информация вы не пишете новый лист, а достаете его из коробки, получаете эту информацию, лист кладете обратно в коробку. Таким образом вы не расходуете много памяти, и делаете программу массовой.

Чтобы что-то положить в коробку нужно произвести операцию присваивания - "=". В одну переменную можно сколько угодно раз присваивать значения, но после присваивания предыдущее значение будет теряться.

Константы

Константы - это переменые которые невозможно изменить, если ей уже присвоено значение. В языке Python нет возможности объявить неизменяемую переменную. Для обозначения переменных, которые никогда не должны изменяться существует договоренность именовать их прописными буквами.

```
MY_CONSTANT = 'моя «константа»'
```

Это не помешает вам ее изменить, но некоторые IDE с анализатором типов, могут обнаруживать замену зачений в таких переменных и указывать, что значение не должно изменяться.

Логические значения (Булев тип)

Несмотря на то, что компьютер хранит для нас текстовую и числовую информацию, в реальности он оперирует лишь двумя значениями: Истина - `True`, Ложь - `False`.

Такой тип данных принято называть булевым. Благодаря этому типу мы можем сравнивать значения между собой, выполнять логические операции, наделять программы вариативностью и много чего еще.

Оператор условия

Одним из важнейших вещей в программировании является оператор условия. Он позволяет задавать ситуации в которых могут потребоваться различные действия при определенных значениях.

Условия могут быть разные: больше, меньше, равно, не равно, вхождение во множество и прочие.

```
if a > b:
  c = a
else:
  c = b
```

В данном случае программа сравнит значения в переменной a и b и запишет в переменную c наибольшее значение.

Выражения + Add a view — Search — New

Оператор	Описание	Вес
lambda	лямбда-выражение	23
or	Логическое "ИЛИ"	22
and	Логическое "И"	21
not x	Логическое "НЕ"	20
in, not in	Проверка принадлежности	19
is, is not	Проверка тождественности	18
<, <=, >, >=, !=, ==	Сравнения	17
|	Побитовое "ИЛИ"	16
^	Побитовое "ИСКЛЮЧИТЕЛЬНО ИЛИ"	15
&	Побитовое "И"	14
<<, >>	Сдвиги	13
+, -	Сложение и вычитание	12
*, /, //, %	Умножение, деление, целочисленное деление и остаток от деления	11
+x, -x	Положительное, отрицательное	10
~x	Побитовое НЕ	9
**	Возведение в степень	8
x.attribute	Ссылка на атрибут	7
x[индекс]	Обращение по индексу	6
x[индекс1:индекс2]	Вырезка	5
f(аргументы ...)	Вызов функции	4
(выражения, ...)	Связка или кортеж [2]	3
[выражения, ...]	Список	2
{ключ:данные, ...}	Словарь	1

+ New

В данной таблице важность операторов идет снизу вверх - чем выше оператор, тем позднее он будет выполнен в операции. Самый простой пример это сложение после умножения.

Несколько условий

```
if a > b and b > c:
  a = b
  b = c
```

Для более удобной читаемости кода следует использовать скобки.

```
if (a > b) and (b > c):
  a = b
  b = c
```

В данном случае b присвоится в a, а c присвоится в b, только тогда когда выполнятся оба условия.

Множественное условие

```
if a < 0:
  s = -1
elif a == 0:
  s = 0
else:
  s = 1
```

В данном блоке кода, если значение в `a` будет больше нуля, то будет выполнена проверка на то, равно ли `a` нулю. Если не равно то в переменную `s` будет присвоено значение `1`, если равно то `0`, а если `a` окажется меньше нуля то — `-1`.

Тест

◄ Проверочное тестирование по лекциям для самокотроля

Перейти на...

Итоговый тест(доступен на сессии) ►