.1_номер компьютера.all.vt

или

вход только в компьютер guest

Введение в язык программирования **Python**

Язык Python прост в изучении и синтаксисе

```java
class HelloWorldApp {
 public static void main(String[] args) {
   System.out.println("Hello World!"); // Display the string.
   }
 }
```

```c
 #include<stdio.h>
 int main(int argc, char** argv)
 {
   printf("Hello World");
  }
```

```
print "Hi there"
```

Python

язык программирования высокого уровня, то есть чтение кода на **Python** и написание кода на нём очень простое — он весьма схож с обычным английским языком;

интерпретируемый — то есть, вам не потребуется специальный компилятор скриптов, чтобы писать на **Python** и запускать его скрипты;

он объектно-ориентированный, то есть — позволяет пользователям управлять структурами данных, называемыми «*объекты*«, для построения и выполнения программ, мы обсудим *объекты* позже, в нашем курсе.;

весёлый в изучении и использовании. **Python** получил своё название в честь **Летающего Цирка Монти Пайтона (*Monty Python's Flying Circus*)**, и примеры в коде и учебники часто ссылаются на это шоу и включают в себя шутки, что бы сделать изучения языка более интересным.

Python. Типы данных

Логический, может принимать одно из двух значений — True (истина) или False (ложь).
Числа, могут быть целыми (1 и 2), с плавающей точкой (1.1 и 1.2), дробными (1/2 и 2/3), и даже комплексными.
Строки — последовательности символов Юникода, например, HTML-документ.
Байты и **массивы байтов**, например, файл изображения в формате JPEG.
Списки — упорядоченные последовательности значений.
Кортежи — упорядоченные неизменяемые последовательности значений.
Множества — неупорядоченные наборы значений.
Словари — неупорядоченные наборы пар вида ключ-значение.

Логический

- Логический тип данных может принимать одно из двух значений: истина или ложь (True/False)
- Результатом вычисления выражений также может быть логическое значение.

```
if size < 0:
    raise ValueError('число должно быть неотрицательным')
```

- Из-за некоторых обстоятельств, связанных с наследием оставшимся от Python 2, логические значения могут трактоваться как числа. True как 1, и False как 0.

Числа

- Python поддерживает как целые числа, так и с плавающей точкой.
- Нет необходимости объявлять тип для их различия; Python определяет его по наличию или отсутствию десятичной точки.
- Можно использовать функцию type() для проверки типа любого значения или переменной.

```
>>> type(1000)
<class 'int'>
```

- Функцию isinstance(переменная, тип) тоже можно использовать для проверки принадлежности значения или переменной определенному типу.

Целые числа и числа с плавающей точкой

- float(целое) – преобразование в число с плавающей точкой
- int(дробь) – преобразоывание дроби в целое отбрасывая дробную часть
- Точность чисел с плавающей точкой равна 15 десятичным знакам в дробной части.
- Целые числа могут быть сколь угодно большими.

Основные операции с числами

- + Сложение двух чисел:

  ```
  print(6 + 2)  # 8
  ```

- - Вычитание двух чисел:

  ```
  print(6 - 2)  # 4
  ```

- * Умножение двух чисел:

  ```
  print(6 * 2)  # 12
  ```

- / Деление двух чисел:

  ```
  print(6 / 2)  # 3.0
  ```

- // Целочисленное деление двух чисел:

  ```
  print(7 / 2)  # 3.5
  print(7 // 2)  # 3
  ```

  Данная операция возвращает целочисленный результат деления

Python. Ввод и вывод данных

print() – команда языка Python, которая выводит то, что в ее скобках на экран.

```
print(1032)
1032
print(2.34)
2.34
print("Hello")
Hello
```

input() - ввод в программу данных с клавиатуры

```
a = input()
hello
print(a)
hello
```

Переменные

Данные хранятся в ячейках памяти компьютера. Когда мы вводим число, оно помещается в какую-то ячейку памяти. Но как потом узнать, куда именно? Как впоследствии обращаться к этим данными? Нужно как-то запомнить, пометить соответствующую ячейку.

Механизм связи между переменными и данными может различаться в зависимости от языка программирования и типов данных. Пока достаточно запомнить, что в программе данные связываются с каким-либо именем и в дальнейшем обращение к ним возможно по этому имени-переменной.

Слово "переменная" обозначает, что сущность может меняться, она непостоянна.

В программе на языке Python, как и на большинстве других языков, связь между данными и переменными устанавливается с помощью знака =. Такая операция называется **присваивание** (также говорят "присвоение").

Имена переменных могут быть любыми. Однако есть несколько общих правил их написания:

- Желательно давать переменным осмысленные имена, говорящие о назначении данных, на которые они ссылаются.
- Имя переменной не должно совпадать с командами языка (зарезервированными ключевыми словами).
- Имя переменной должно начинаться с буквы или символа подчеркивания (_), но не с цифры.
- Имя переменной не должно содержать пробелы.

В этом примере используются четыре переменные:

переменная **a** хранит значение типа **int** (целое число),
переменная **b** — типа **float** (действительное число),
переменная **c** — типа **str** (строка),
переменная **d** — типа **list** (список, в данном случае из трех целых чисел).

```
a = 10
b = 3.1415926
c = "Hello"
d = [1, 2, 3]
```

Python — язык с динамической типизацией: каждая переменная в каждый момент времени имеет определенный тип, но этот тип может меняться по ходу выполнения программы, достаточно просто присвоить ей новое значение другого типа.

Строки

Операция	Пример кода

Конкатенация (сложение)	>>> S1 = 'spam'
	>>> S2 = 'eggs'
	>>> print(S1 + S2)
	'spameggs'

Дублирование строки	>>> print('spam' * 3) spamspamspam
Длина строки	>>> len('spam') 4

Доступ по индексу	>>> S = 'spam'
	>>> S[0]
	's'

Извлечение среза	>>> s = 'spameggs'
	>>> s[3:5]
	'me'

Списки

```
>>> a_list = ['a', 'b', ‘c', ‘d', ‘e']
>>> a_list[1:3]
['b', ‘c']
>>> a_list += [‘f’]
['a', 'b', ‘c', ‘d', ‘e‘, ‘f’]
>>> a_list.append(‘g’)
['a', 'b', ‘c', ‘d', ‘e‘, ‘f’, ‘g’]
>>> a_list.extend([‘h’, ‘I’])
['a', 'b', ‘c', ‘d', ‘e‘, ‘f’, ‘g’, ‘h’, ‘I’]
>>> a_list.insert(0, ‘a’)
['a', 'a', 'b', ‘c', ‘d', ‘e‘, ‘f’, ‘g’, ‘h’, ‘I’]
>>> a_list.count(‘a’)
0, 1
>>> a_list.index(‘a’)
0
```

```
# Объявление списка
# Срез со второго по третий символ

# Добавление элемента в список

# Еще один вариант добавления элементов

# И еще один

# Добавление элемента в указанную
позицию
# Индексы вхождений элемента в список

#Индекс первого вхождения в список
```

Поиск в списке

```
>>> 'j' in a_list
False
>>> len(a_list)
10
>>> a_list.remove('a')
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'I']
>>> a_list.pop([1])
'b'
>>> a_list.reverse()
['i', 'h', 'g', 'f', 'e', 'd', 'c', 'b', 'a']
>>> a_list.copy()
['i', 'h', 'g', 'f', 'e', 'd', 'c', 'b', 'a']
>>> a_list.clear()
[]
```

```
# Проверка на вхождение элемента в список
# Длинна списка

# Удаление элемента из списка

# Выдергиваем элемент из списка

# Отзеркаливание списка

# Копирование списка

# Очищение списка
```

Результат операции сложения можно присвоить другой переменной…

```
>>> a = 3
>>> b = 2
>>> c = a + b
>>> print(c)
5
```

Результат операции сложения можно присвоить самой переменной, в таком случае можно использовать полную или сокращенную запись, полная выглядит так

```
>>> a = 3
>>> b = 2
>>> a = a + b
>>> print(a)
5
```

сокращенная
так

```
>>> a = 3
>>> b = 2
>>> a += b
>>> print(a)
5
```

Все перечисленные варианты использования операции сложения могут быть применены для всех нижеследующих операций.

```
>>> 4-2
2
>>> a = 5
>>> b = 7
>>> a - b
-2
```

Умножение.

```
>>> 5 * 8
40
>>> a = 4
>>> a *= 10
>>> print(a)
40
```

Деление.

```
>>> 9 / 3
3.0
>>> a = 7
>>> b = 4
>>> a / b
1.75
```

Получение целой части от деления.

```
>>> 9 // 3
3
>>> a = 7
>>> b = 4
>>> a // b
1
```

Получение остатка от деления.

```
>>> 9 % 5
4
>>> a = 7
>>> b = 4
>>> a % b
3
```

Возведение в степень.

```
>>> 5 ** 4
625
>>> a = 4
>>> b = 3
>>> a ** b
64
```

ОПЕРАЦИИ В ПРОГРАММИРОВАНИИ

```
>>> 10.25 + 98.36
108.61
>>> 'Hello' + 'World'
'HelloWorld'
```

Изменение типов данных

Эти функции преобразуют то, что помещается в их скобки соответственно в целое число, вещественное число или строку.

```
>>> str(1) + 'a'
'1a'
>>> int('3') + 4
7
```

Сложение строк:

```
>>> str(4) + str(1.2)
'41.2'
```

Обмен значений переменных

Поскольку в Python есть такая вещь как множественное присваивание, то обмен значений переменных можно выполнить в одну строчку:

Таким образом классический алгоритм обмена значений двух переменных выглядит так:

```
a = 5
b = 6

buf = a
a = b
b = buf
```

Поскольку в Python есть такая вещь как множественное присваивание, то обмен значений переменных можно выполнить в одну строчку:

```
>>> a = 10
>>> b = 20
>>> a, b = b, a
>>> a
20
>>> b
10
```

Практическая работа

Пользователь вводит два числа. Найдите сумму и произведение данных чисел.

Практическая работа

Пользователь вводит три числа.

Увеличьте первое число в два раза, второе числа уменьшите на 3, третье число возведите в квадрат и затем найдите сумму новых трех чисел.

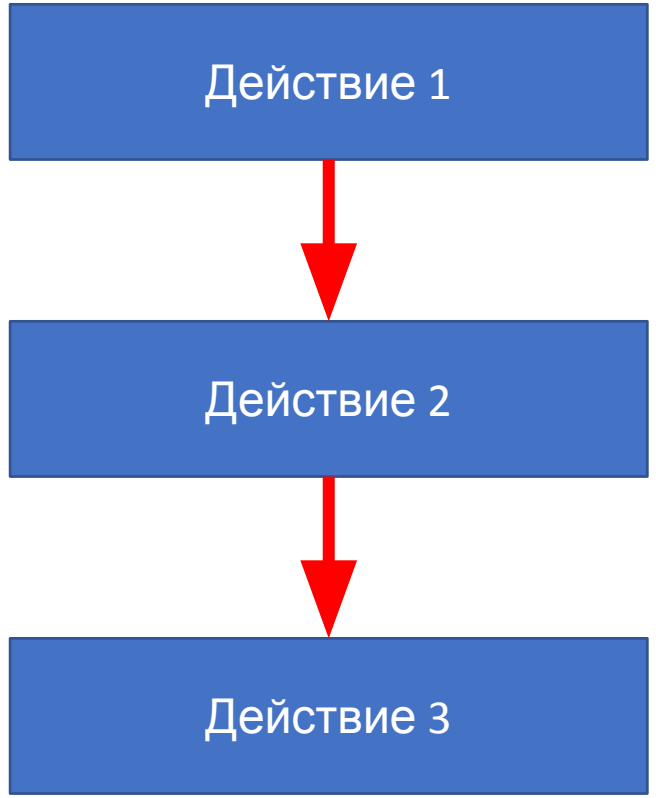
Практическая работа

Объявить переменные с помощью которых можно будет посчитать общую сумму покупки нескольких товаров.

Например плитки шоколада, кофе и пакеты молока.

Ветвление
Условный оператор

Ход выполнения программы может быть *линейным*, то есть таким, когда выражения выполняются друг за другом, начиная с первого и заканчивая последним. Ни одна строка кода программы не пропускается.



Однако чаще в программах бывает не так. При выполнении кода, в зависимости от тех или иных условий, некоторые его участки могут быть опущены, в то время как другие – выполнены. Иными словами, в программе может присутствовать *ветвление*, которое реализуется **условным оператором – особой конструкцией языка программирования**.

```
if логическое_выражение {
    выражение 1;
    выражение 2;
    …
}
```

Перевести на человеческий язык можно так: **если логическое выражение возвращает истину, то выполняются выражения внутри фигурных скобок**; если логическое выражение возвращает ложь, то код внутри фигурных скобок не выполняется. С английского "if" переводится как "если".

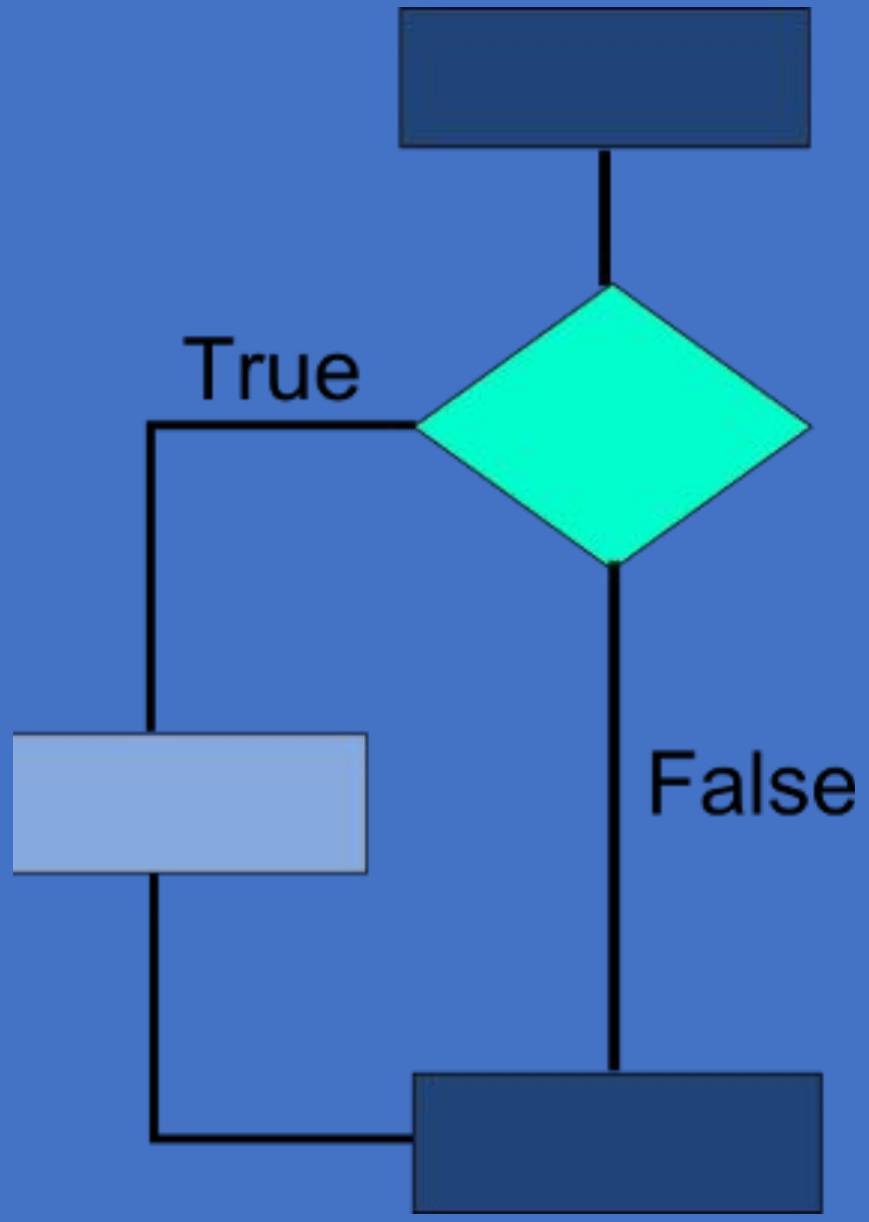
```
if логическое_выражение {
    выражение 1;
    выражение 2;
    …
}
```

Конструкция if логическое_выражение называется **заголовком условного оператора**. Выражения внутри фигурных скобок – **телом условного оператора**. Тело может содержать как множество выражений, так и всего одно или даже быть пустым.

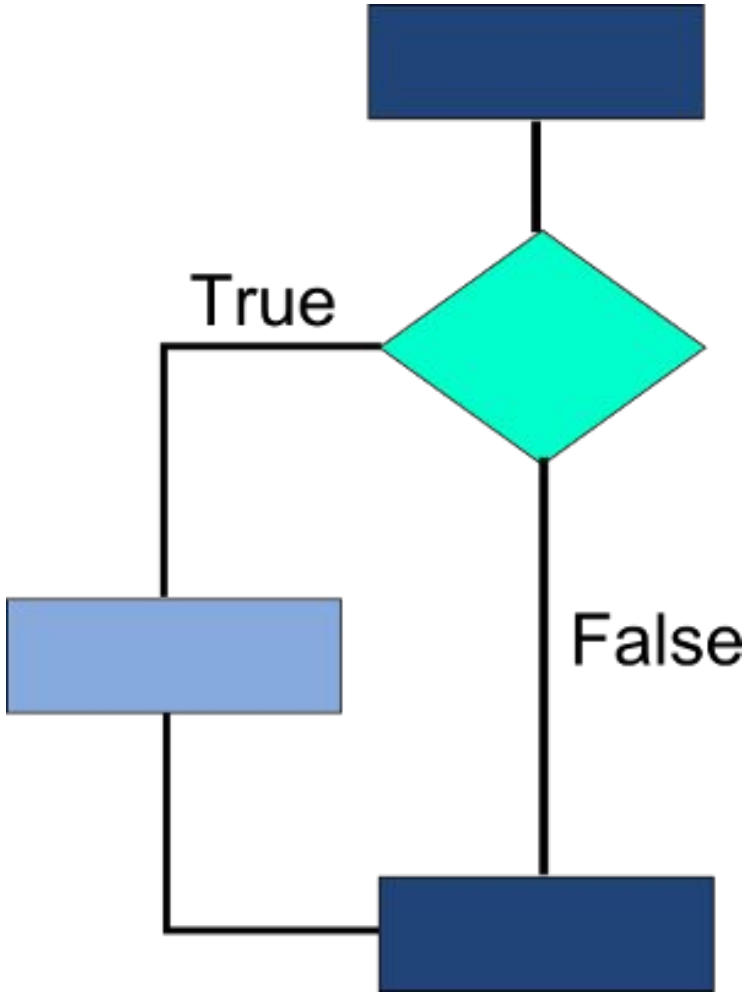
Пример использования условного оператора в языке программирования Python:

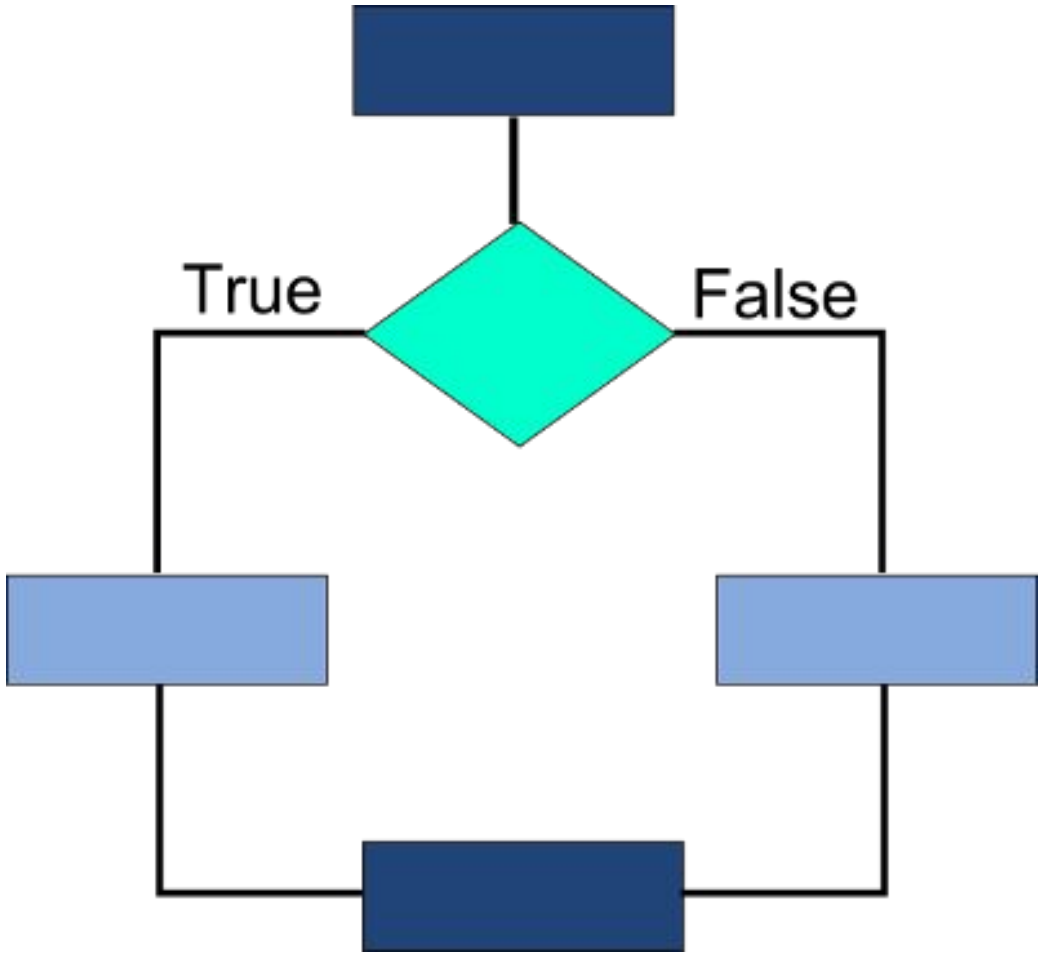
```
if n < 100:
    b = n + a
```

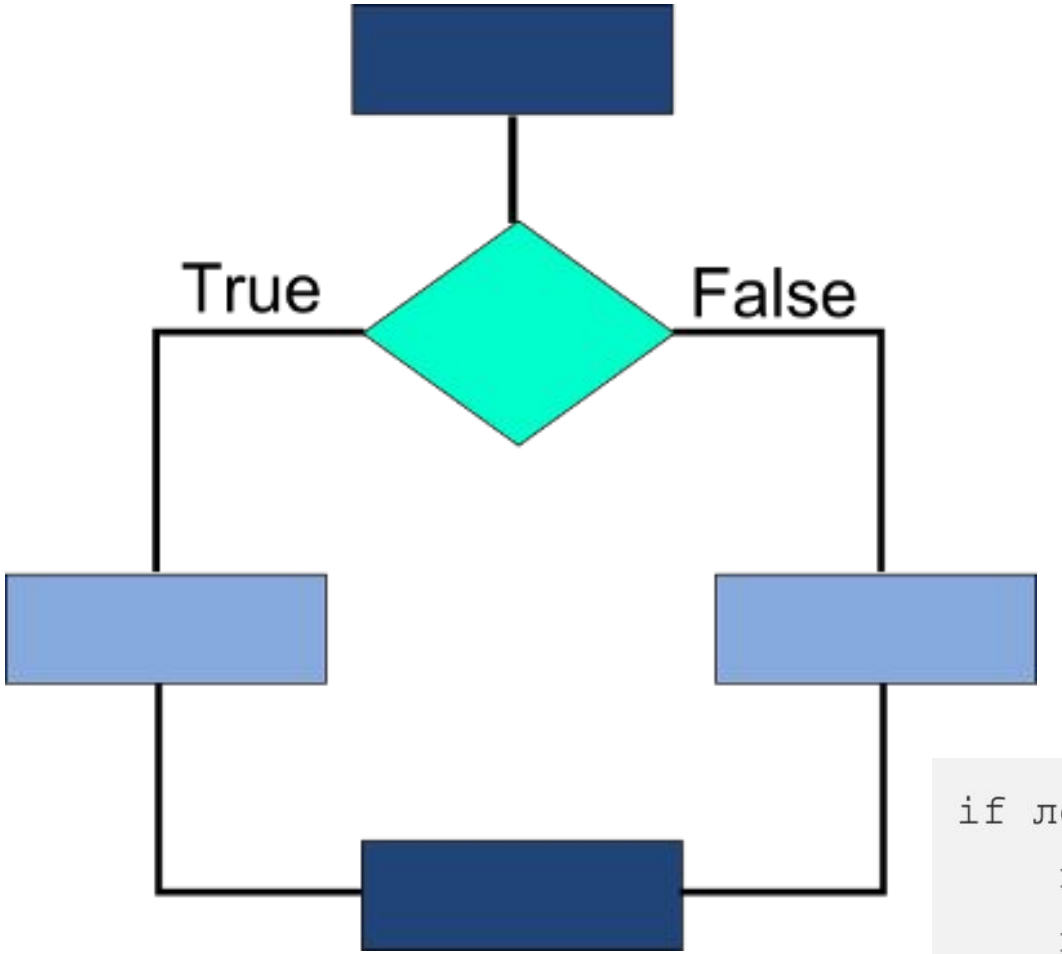
Для небольших программ иногда чертят так называемые блок-схемы, отражающие алгоритм выполнения. В языке блок-схем определенные конструкции обозначаются своими фигурами. Так блок действий обозначается прямоугольником, а логическое выражение – ромбом. Для кода выше блок-схема может выглядеть так:



Условный оператор может включать не одну ветку, а две, реализуя тем самым полноценное ветвление.








```
if логическое_выражение {
    выражение 1;
    выражение 2;
    …
}
else {
    выражение 3;
    …
}
```

Если условие при инструкции if оказывается ложным, то выполняется блок кода при инструкции else. **Ситуация, при которой бы выполнились обе ветви, невозможна.** Либо код, принадлежащий if, либо код, принадлежащий else. Никак иначе. **В заголовке else никогда не бывает логического выражения.**

Пример кода с веткой else на языке программирования Python:

```
tovar1 = 50
tovar2 = 32
if tovar1+ tovar2 > 99 :
    print("99 рублей недостаточно")
else:
    print("Чек оплачен")
```

Множественное ветвление: **if-elif-else**

```
old = int(input('Ваш возраст: '))

print('Рекомендовано:', end=' ')

if 3 <= old < 6:
    print('"Заяц в лабиринте"')

if 6 <= old < 12:
    print('"Марсианин"')

if 12 <= old < 16:
    print('"Загадочный остров"')

if 16 <= old:
    print('"Поток сознания"')
```

Следующий if никак не связан с предыдущим. Ответом является вложение условных операторов друг в друга:

Рассмотрим поток выполнения этого варианта кода. Сначала проверяется условие в первом if (он же самый внешний). Если здесь было получено True, то тело этого if выполняется, а в ветку else мы даже не заходим, так как она срабатывает только тогда, когда в условии if возникает ложь.

```
old = int(input('Ваш возраст: '))

print('Рекомендовано:', end=' ')

if 3 <= old < 6:
    print('"Заяц в лабиринте"')
else:
    if 6 <= old < 12:
        print('"Марсианин"')
    else:
        if 12 <= old < 16:
            print('"Загадочный остров"')
        else:
            if 16 <= old:
                print('"Поток сознания"')
```

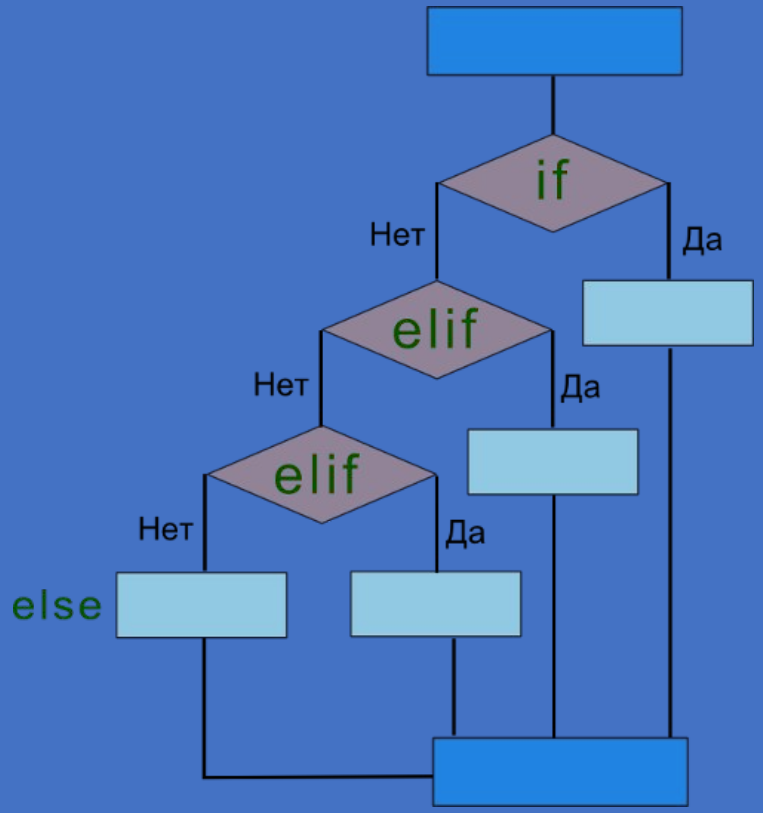
Теперь зададимся следующим вопросом. Можно ли как-то оптимизировать код множественного ветвления и не строить лестницу из вложенных друг в друга условных операторов? Во многих языках программирования, где отступы используются только для удобства чтения программистом, но не имеют никакого синтаксического значения, часто используется подобный стиль:

```
if логическое_выражение {
    … ;
}
else if логическое_выражение {
    … ;
}
else if логическое_выражение {
    … ;
}
else {
    … ;
}
```

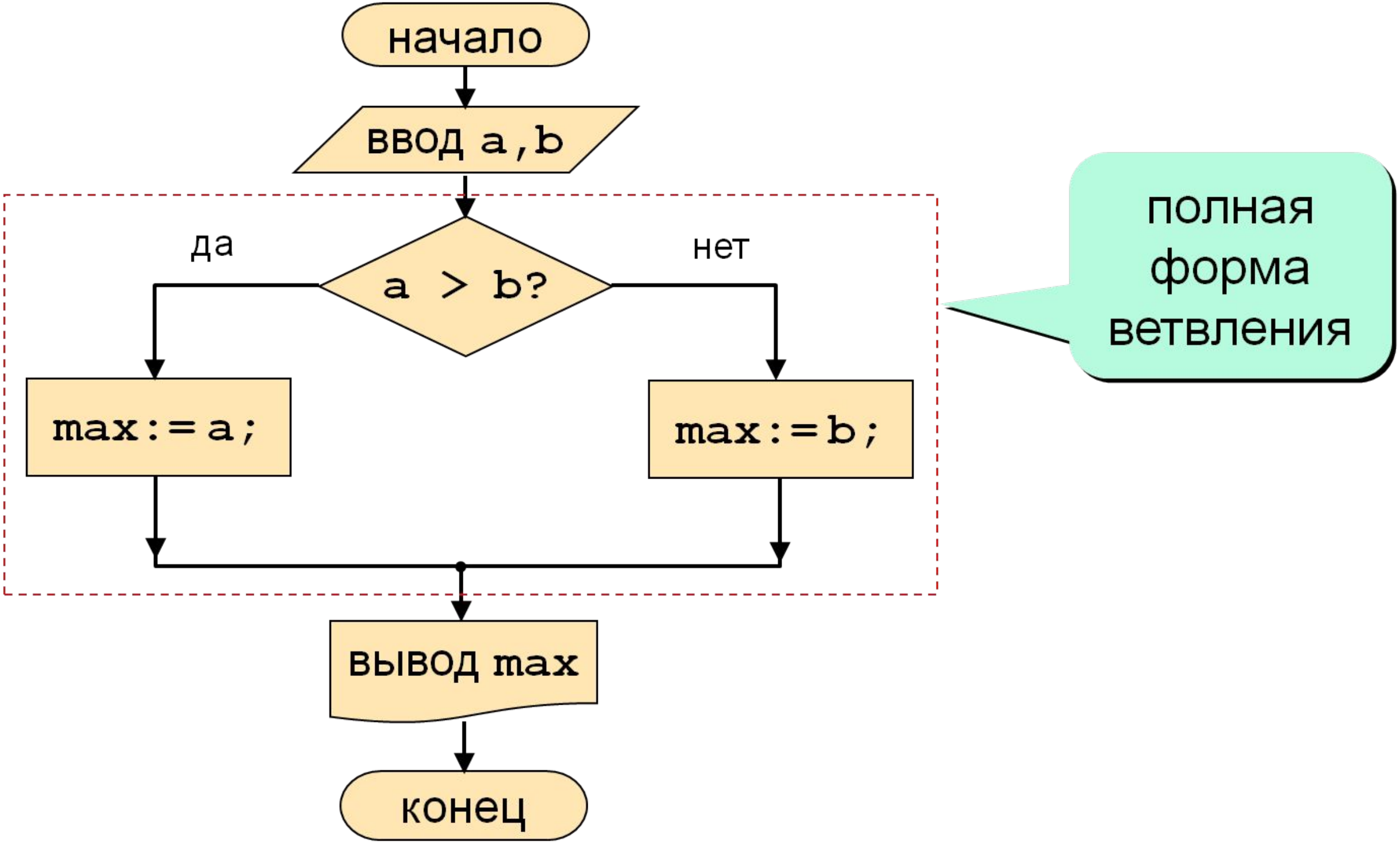
Слово "elif" образовано от двух первых букв слова "else", к которым присоединено слово "if". Это можно перевести как "иначе если".
В отличие от else, **в заголовке elif обязательно должно быть логическое выражение** также, как в заголовке if. Перепишем нашу программу, используя конструкцию множественного ветвления:

```
old = int(input('Ваш возраст: '))

print('Рекомендовано:', end=' ')

if 3 <= old < 6:
    print('"Заяц в лабиринте"')
elif 6 <= old < 12:
    print('"Марсианин"')
elif 12 <= old < 16:
    print('"Загадочный остров"')
elif 16 <= old:
    print('"Поток сознания"')
```

Обратите внимание, в конце, после всех elif, может использоваться одна ветка else для обработки случаев, не попавших в условия ветки if и всех elif. Блок-схему полной конструкции if-elif-…-elif-else можно изобразить так:



Как только тело if или какого-нибудь elif выполняется, программа сразу же возвращается в основную ветку (нижний ярко-голубой прямоугольник), а все нижеследующие elif, а также else пропускаются.


начало
ввод a,b
a > b?
да
нет
max:=a;
max:=b;
вывод max
конец
полная форма ветвления

Напишите программу по следующему описанию:

a. двум переменным присваиваются числовые значения;

b. если значение первой переменной больше второй, то найти разницу значений переменных (вычесть из первой вторую), результат связать с третьей переменной;

c. если первая переменная имеет меньшее значение, чем вторая, то третью переменную связать с результатом суммы значений двух первых переменных;

d. во всех остальных случаях, присвоить третьей переменной значение первой переменной;

e. вывести значение третьей переменной на экран.

Определить положительное или отрицательное число

Пользователь вводит число. Необходимо проверить оно больше нуля? **Если** да, то выводим об этом сообщение, **иначе если** оно меньше нуля, то выводим сообщением об этом, **иначе** оно равно нулю и программа выводит сообщение об этом.

Найти максимальное число из трех

Вводятся три целых числа. Определить какое из них наибольшее.

Пусть *a*, *b*, *c* - переменные, которым присваиваются введенные числа, а переменная *m* в конечном итоге должна будет содержать значение наибольшей переменной. Тогда алгоритм программы сведется к следующему:

1. Сначала предположим, что переменная *a* содержит наибольшее значение. Присвоим его переменной *m*.

2. Если текущее значение *m* меньше, чем у *b*, то следует присвоить *m* значение *b*. Если это не так, то не изменять значение *m*.

3. Если текущее значение *m* меньше, чем у *c*, то присвоить *m* значение *c*. Иначе ничего не делать.

```
a = int(input('Введите число a='))
b = int(input('Введите число b='))
c = int(input('Введите число c='))

m = a
if m < b:
    m = b
if m < c:
    m = c

print(m)
```

на другое

Вводятся два целых числа. Проверить делится ли первое на второе. Вывести на экран сообщение об этом, а также остаток (если он есть) и частное (в любом случае).

1) Если первое число нацело делится на второе, то вывести сообщение об этом.

2) Иначе вывести сообщение о том, что первое число не делится на второе, найти остаток от деления и также вывести его.

3) В конце программы найти частное от деления чисел и вывести его.

Даны два ненулевых числа.

Определите, совпадают ли у них знаки или нет.

Ввести число с клавиатуры проверить его на четность.

Используя оператор выбора, составьте программу, которая по введенному номеру месяца будет выводить название соответствующего времени года (зима, весна, лето, осень).

Логического выражения и логический тип данных

На прошлом уроке были описаны три типа данных: целые, дробные числа, а также строки. Также выделяют *логический тип данных*. У этого типа всего два возможных значения: **True** (правда) — 1 и **False** (ложь) — 0. Только эти значения могут быть результатом логических выражений.

Примеры работы с логическими выражениями на языке программирования Python (после # написаны комментарии):

```
x == 4 # x равен 4
x == 7 # x равен 7
x != 7 # x не равен 7
x != 4 # x не равен 4
x > 5 # x больше 5
x < 5 # x меньше 5
x >= 6 # x больше или равен 6
x <= 6 # x меньше или равен 6
```

Сложные логические выражения

Может понадобиться получить ответа "Да" или "Нет" в зависимости от результата выполнения двух простых выражений. Например, *"на улице идет снег или дождь"*, *"переменная new больше 12 и меньше 20"* и т.п.

В таких случаях используются специальные операторы, объединяющие два и более простых логических выражения. Широко используются два способа объединения: через, так называемые, логические И (and) и ИЛИ (or).

Чтобы получить истину (**True**) при использовании оператора **and**, необходимо, чтобы результаты обоих простых выражений, которые связывает данный оператор, были истинными. Если хотя бы в одном случае результатом будет **False** (ложь), то и все сложное выражение будет ложным.

Чтобы получить истину (**True**) при использовании оператора **or**, необходимо, чтобы результаты хотя бы одного простого выражения, входящего в состав сложного, был истинным. В случае оператора **or** сложное выражение становится ложным лишь тогда, когда ложны все составляющие его простые выражения.

Примеры работы со сложными логическими выражениями на языке программирования Python (после # написаны комментарии):

```
x =8
y = 13
x == 8 and y < 15 # x равен 8 и y меньше 15
x > 8 and y < 15 # x больше 8 и y меньше 15
x != 0 or y >15 # x не равен 0 или y меньше 15
x < 0 or y >15 # x меньше 0 или y меньше 15
```

Практическая работа

1) Присвойте двум переменным любые числовые значения.

2) Составьте четыре сложных логических выражения с помощью оператора **and**, два из которых должны давать истину, а два других - ложь.

3) Аналогично выполните п. 2, но уже используя оператор **or**.

4) Попробуйте использовать в сложных логических выражениях работу с переменными строкового типа.

В компьютер вводятся два числа. Если первое больше второго, то вычислить их сумму, иначе - произведение. После этого компьютер должен напечатать результат и текст ЗАДАЧА РЕШЕНА

Человек вводит в компьютер число. Если оно находится в интервале от 28 до 30, то нужно напечатать текст ПОПАЛ, если оно больше или равно 30 - то ПЕРЕЛЕТ, если оно находится на отрезке от 0 до 28, то НЕДОЛЕТ, если число меньше нуля — НЕ БЕЙ ПО СВОИМ

В три переменные a, b и c явно записаны программистом три целых попарно неравных между собой числа.

Создать программу, которая переставит числа в переменных таким образом, чтобы при выводе на экран последовательность a, b и c оказалась строго возрастающей.

Среди трех чисел найти среднее

Вводятся три разных числа. Найти, какое из них является средним (больше одного, но меньше другого).

Проверить, лежит ли первое число между двумя другими. При этом может быть два случая:
-первое больше второго и первое меньше третьего,
-первое меньше второго и первое больше третьего.

Если ни один из вариантов не вернул истину, значит первое число не среднее. Тогда проверяется, не лежит ли второе число между двумя другими. Это может быть в двух случаях, когда
-второе больше первого и меньше третьего,
-второе меньше первого и больше третьего.

Если эти варианты также не вернули истину, то остается только один вариант - посередине лежит третье число.

Среди трех чисел найти среднее

```
print('Введите три числа: ')
a = int(input())
b = int(input())
c = int(input())

if b < a < c or c < a < b:
    print('Среднее:', a)
elif a < b < c or c < b < a:
    print('Среднее:', b)
else:
    print('Среднее:', c)
```

Пользователь вводит целое число.

Программа должна ответить, четным или нечетным является это число, делится ли оно на 3 и делится ли оно на 6.

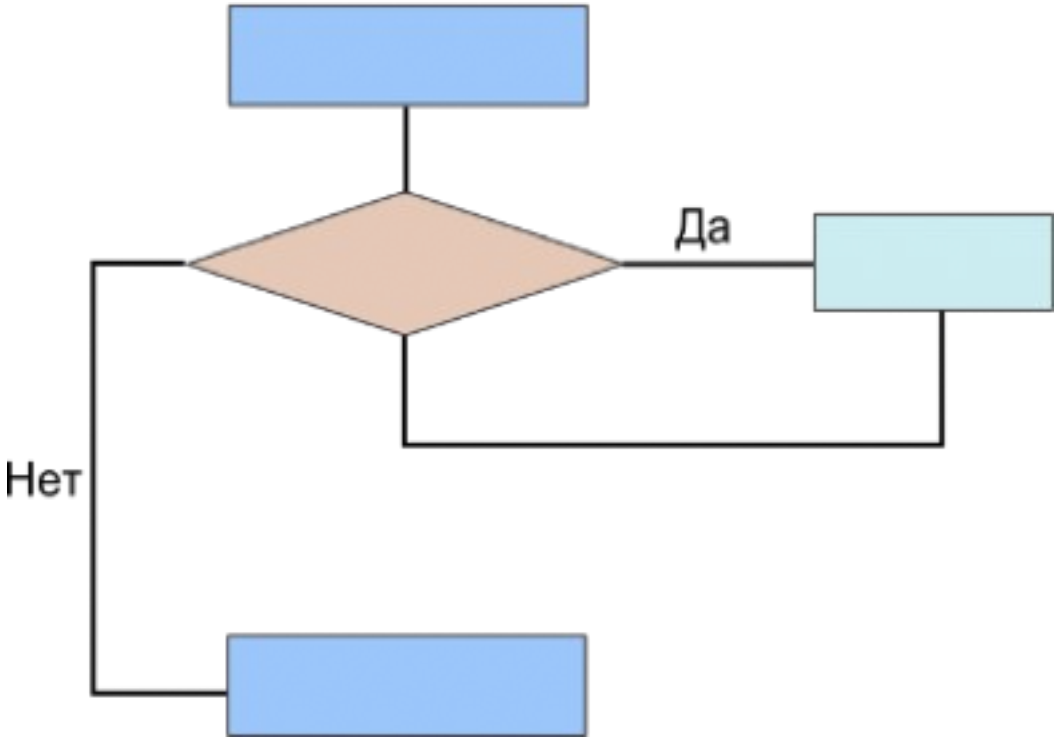
Циклы

Циклы — это инструкции, выполняющие одну и туже последовательность действий, пока действует заданное условие.

Цикл while

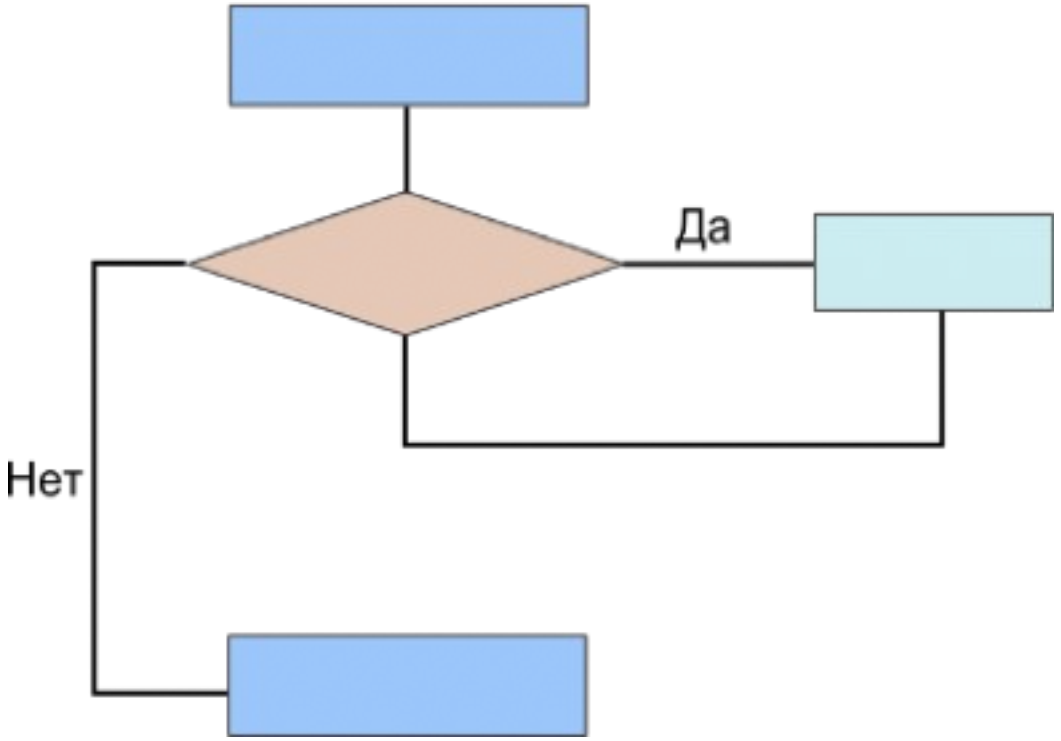
Универсальным организатором цикла в языке программирования Python (как и во многих других языках) является конструкция **while**. Слово "while" с английского языка переводится как "пока" ("пока логическое выражение возвращает истину, выполнять определенные операции"). Конструкцию **while** на языке Python можно описать следующей схемой:

```
while a логический_оператор b:
    действие(я)
    изменение a
```



в заголовке цикла **while**, обычно называют *счетчиком*. Как и всякой переменной ей можно давать произвольные имена, однако очень часто используют буквы i и j. Простейший цикл на языке программирования Python может выглядеть так:

```
str1 = "+"
i=0
while i < 10:
    print (str1)
    i=i+1
```



Более сложный пример с использованием цикла:

```
fib1 = 0
fib2 = 1
print (fib1)
print (fib2)
n = 10
i=0
while i < n:
    fib_sum = fib1 + fib2
    print (fib_sum)
        fib1 = fib2
        fib2 = fib_sum
        i=i+1
```

Этот пример выводит ***числа Фибоначчи*** — ряд чисел, в котором каждое последующее число равно сумме двух предыдущих: 0, 1, 1, 2, 3, 5, 8, 13 и т.д. Скрипт выводит двенадцать членов ряда: два (0 и 1) выводятся вне цикла и десять выводятся в результате выполнения цикла.

Практическая работа

1. Напишите цикл, выводящий ряд четных чисел от 0 до 20.

2. Затем, каждое третье число в ряде от -1 до -21.

3. Самостоятельно придумайте программу на Python, в которой бы использовался цикл **while**.

Ввод данных с клавиатуры

Практическая работа

Создайте скрипт, который бы **запрашивал** у пользователя

- его имя: "Как тебя зовут? "
- возраст: "Сколько тебе лет? "
- место жительства: "Где ты проживаешь? "

Затем **выводил** три строки

- "This is *имя*
- "It is *возраст* "
- "He live in *место_жительства*"

где вместо *имя, возраст, место_жительства* должны быть соответствующие данные, введенные пользователем.

Практическая работа

Напишите программу, которая предлагала бы пользователю решить пример **4*100-54**.

Если пользователь напишет правильный ответ, то получит поздравление от программы, иначе – программа сообщит ему об ошибке.

(При решении задачи используйте конструкцию **if-else**.)

Практическая работа

Перепишите предыдущую программу так, чтобы пользователю предлагалось решать пример до тех пор, пока он не напишет правильный ответ.

(При решении задачи используйте цикл **while**.)

Строки

Строка — это сложный тип данных, представляющий собой последовательность символов.

Строки в языке программирования Python могут заключаться как в одиночные, так и двойные кавычки. Однако, начало и конец строки должны обрамляться одинаковым типом кавычек.

Существует специальная функция **len()**, позволяющая измерить длину строки. Результатом выполнения данной функции является число, показывающее количество символов в строке.

Также для строк существуют операции *конкатенации* (+) и *дублирования* (*).

```
print(len('It is a long string’) )

'!!!' + ' Hello World ' + '!!!'

'-' * 20
```

В последовательностях важен порядок символов, у каждого символа в строке есть уникальный порядковый номер — ***индекс***.

Можно обращаться к конкретному символу в строке и извлекать его с помощью ***оператора индексирования***, который представляет собой квадратные скобки с номером символа в них.

```
s = 'STRING’
```

Что бы извлечь символы 'S', 'R' и 'G' мы можем выполнить следующие простые операции:

```
print(s[0], s[2], s[5])
```

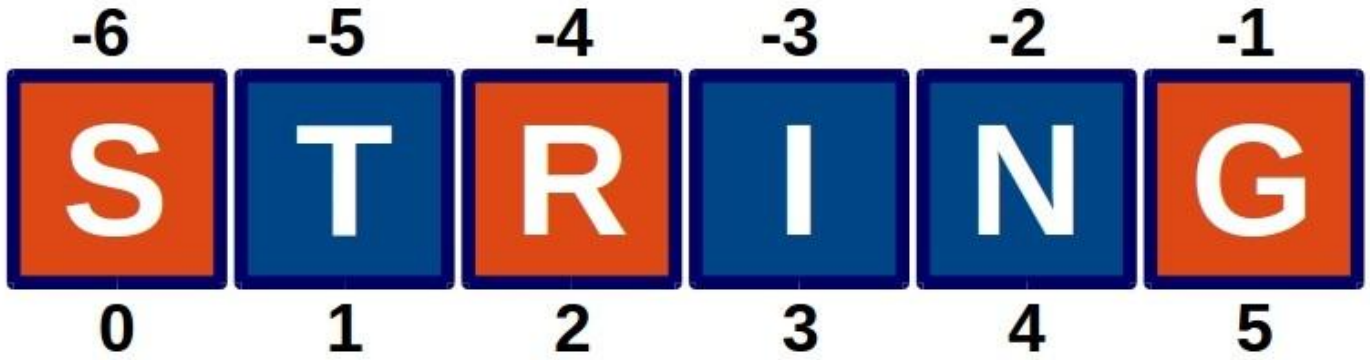
В последовательностях важен порядок символов, у каждого символа в строке есть уникальный порядковый номер — ***индекс***.

Можно обращаться к конкретному символу в строке и извлекать его с помощью ***оператора индексирования***, который представляет собой квадратные скобки с номером символа в них.

```
s = 'STRING’
```

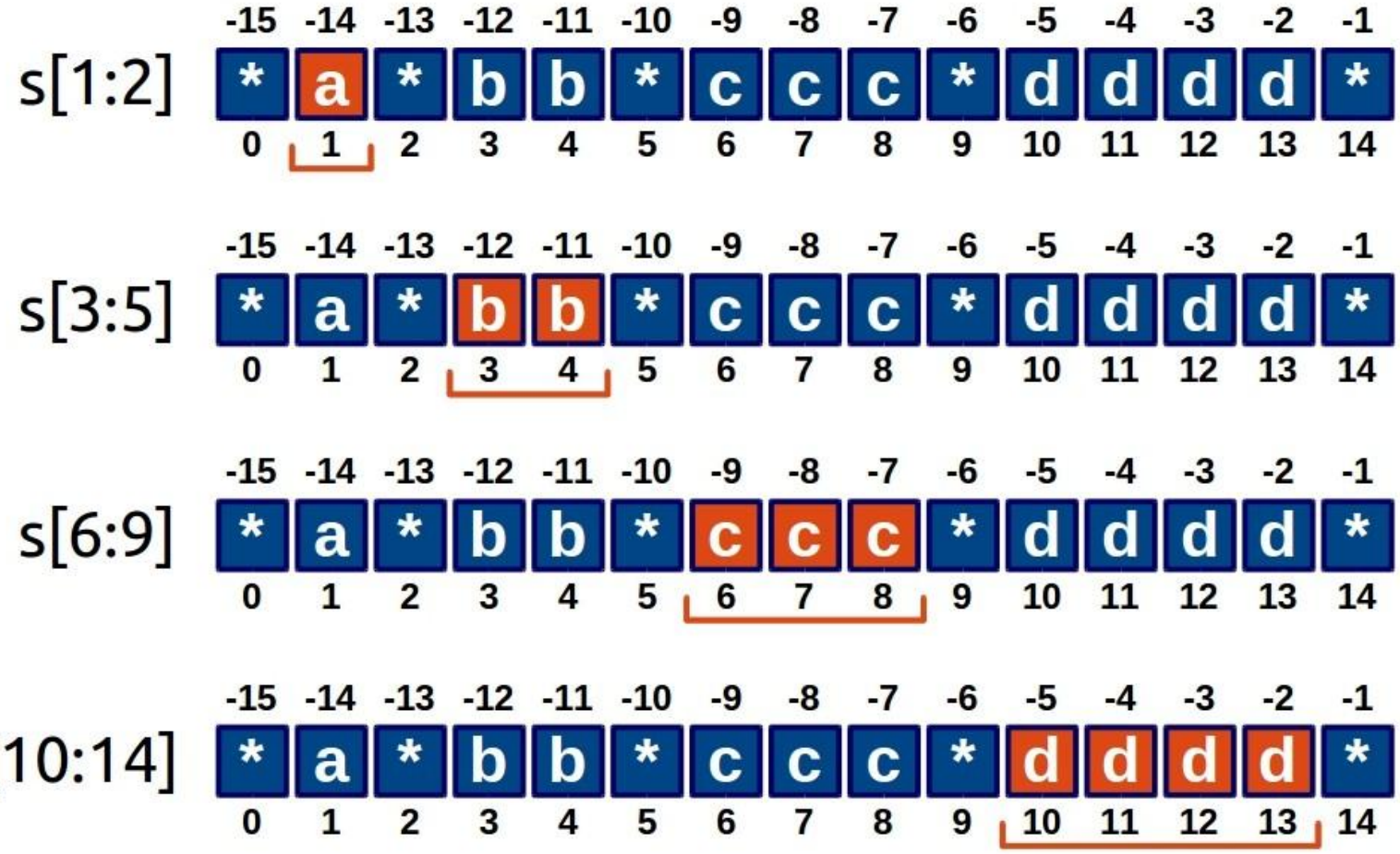
Индексы могут быть отрицательными, т.е. отсчитываться от конца строки, для извлечения тех же символов мы можем выполнить и такие операции:

```
s[-6], s[-4], s[-1]
```



(*подстроку*). Оператор извлечения среза из строки выглядит так: [X:Y). X – это индекс начала среза, а Y – его окончания; причем символ с номером Y в срез уже не входит.

Если отсутствует первый индекс, то срез берется от начала до второго индекса; при отсутствии второго индекса, срез берется от первого индекса до конца строки.



Практическая работа

Свяжите переменную с любой строкой, состоящей не менее чем **из 8 символов**.

Извлеките из строки:
-первый символ
-затем последний
-третий с начала
-третий с конца
Измерьте длину вашей строки

Практическая работа

Присвойте произвольную строку длиной 10-15 символов переменной и извлеките из нее следующие срезы:

- первые восемь символов
- четыре символа из центра строки
- символы с индексами кратными трем.

Списки — изменяемые последовательности

Списки в языке программирования Python, как и строки, являются упорядоченными последовательностями.

Однако, в отличии от строк, списки состоят не из символов, а из различных объектов (значений, данных), и заключаются не в кавычки, а в квадратные скобки [ ]. Объекты отделяются друг от друга с помощью запятой.

Списки могут состоять из различных объектов: чисел, строк и даже других списков. В последнем случае, списки называют **вложенными**.

```
a =[23, 656, -20, 67, -45] # список целых чисел
b = [4.15, 5.93, 6.45, 9.3, 10.0, 11.6] # список из дробных чисел
c =  ["Katy", "Sergei", "Oleg", "Dasha"] # список из строк
d = ["Москва", "Титова", 12, 148] # смешанный список
e =[[0, 0, 0], [0, 0, 1], [0, 1, 0]] # список, состоящий из списков
```

Как и над строками над списками можно выполнять операции соединения и повторения:

```
c= [45, -12, 'april'] + [21, 48.5, 33]
print(c)

d=[[0,0],[0,1],[1,1]] * 3
print(d)
```

По аналогии с символами строк, можно получать доступ к объектам списка по их индексам, извлекать срезы, измерять длину списка:

```
ee = ['a','b','c','d','e','f’]


print(len(ee))
print(ee[0])
print(ee[4])
print(ee[0:3])
print(ee[3:])
```

Практическая работа

1-Создайте два любых списка и свяжите их с переменными.

2-Извлеките из первого списка второй элемент.

3-**Измените во втором списке последний объект. Выведите список на экран.**

4-Соедините оба списка в один, присвоив результат новой переменной. Выведите получившийся список на экран.

5-Выведите срез из соединенного списка так, чтобы туда попали **некоторые части обоих первых списков**. Срез свяжите с очередной новой переменной. Выведите значение этой переменной.

6-Добавьте в список-срез (d) два новых элемента и снова выведите его.

Пример выполнения практической работы

1
```
>>> spisok1 = [45, 2, 8, 97, 34]
>>> spisok2 = [65, 23, 10]
```

2
```
>>> spisok1 [1]
2
```

3
```
>>> spisok2 [-1] = 12
>>> spisok2
[65, 23, 12]
```

4
```
>>> big_spisok = spisok1 + spisok2
>>> big_spisok
[45, 2, 8, 97, 34, 65, 23, 12]
```

5
```
>>> small_spisok = big_spisok [3:7]
>>> small_spisok
[97, 34, 65, 23]
```

6
```
>>> small_spisok [4:6] = [56, 84]
>>> small_spisok
[97, 34, 65, 23, 56, 84]
```

Введение в словари

Словари

Одним из сложных типов данных (наряду со строками и списками) в языке программирования Python являются словари.

*Словарь - это **изменяемый** (как список) **неупорядоченный** (в отличие от строк и списков) набор пар "ключ:значение".*

провести аналогию
с обычным словарем, например, англо-русским.

На каждое английское слово в таком словаре есть русское слово-перевод: cat – кошка,
dog – собака, table – стол и т.д.

Если англо-русский словарь описывать с помощью Python, то английские слова будут ключами, а русские — их значениями:

```
{'cat':'кошка', 'dog':'собака', 'bird':'птица',
'mouse':'мышь’}
```

Синтаксис словаря на Питоне:

```
{ключ : значение, ключ : значение, ключ : значение, ...}
```

Цикл for

цикл **for**, который представляет собой цикл обхода заданного множества элементов (символов строки, объектов списка или словаря) и выполнения в своем теле различных операций над ними. Например, если имеется список чисел, и необходимо увеличить значение каждого элемента на две единицы, то можно перебрать список с помощью цикла **for**, выполнив над каждым его элементом соответствующее действие.

```
spisok =
[0,10,20,30,40,50,60,70,80,90]
i = 0
for a in spisok:
    spisok[i] = a + 2
    i=i+1
print(spisok)
```

Перебирать можно и строки, если не пытаться их при этом изменять:

```
stroka = "привет"
for bukva in stroka:
  print(bukva, end=' * ‘)
```

```
п * р * и* в * е * т *
```

Цикл **for** используется и для работы со словарями:

```
d =
{1:'one',2:'two',3:'three',4:'four'}
for key in d:
  d[key] = d[key] + '!'

{1: 'one!', 2: 'two!', 3: 'three!', 4:
'four!'} >>>
```

Цикл FOR

```
for x in range(1,11):
    print ( 2**x )

# 2 4 8 16 ... 1024
```

```
for i in range(3):
    print(i)
# 1
# 2
# 3
```

Шаг счетчика цикла можно менять:

```
for x in range(1,11,2):
   print ( 2**x )
```

Отрицательный шаг:

```
for i in range(10, 7, -1):
    print(i)
# 10
# 9
# 8
```

Еще пример работы:

```
for i in 'hello world':
    if i == 'o':
        continue
    print(i, end='')

# hell  wrld
```

Оператор continue в **Python** возвращает выполнение кода к началу цикла при срабатывании заданного условия как истина (true).

Практическая работа

Создайте список, состоящий из четырех строк.
Затем, с помощью цикла **for**, выведите строки поочередно на экран.

Практическая работа

Измените предыдущую программу так, чтобы в конце каждой строки добавлялось тире (‘-’). (Подсказка: цикл **for** может быть вложен в другой цикл.)

Пример выполнения практической работы

```
# задание 1
list1 = ['hi','hello','good morning','how do you do']
for i in list1:
    print(i)
```

Пример выполнения практической работы

```
# задание 2
list1 = ['hi','hello','good morning','how do you do']
for i in list1:
    for j in i:
        print(j,end='-')
        print()
```

Практическая работа

Вывести на экран все чётные значения в диапазоне от 1 до 497

Практическая работа (Решение)

Вывести на экран все чётные значения в диапазоне от 1 до 497

```
for i in range(1, 498):
    if i%2==0:
    print(i)
```

Практическая работа

Посчитать сумму числового ряда от 0 до 14 включительно. Например, 0+1+2+3+…+14

Практическая работа (Решение)

Посчитать сумму числового ряда от 0 до 14 включительно. Например, 0+1+2+3+…+14

```
s=0
for i in range(0,15):
    s=i + s
print (s)
```