ЦЕНТР ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

Программирование на языке Python. Уровень 1.

Управляющие конструкции. Ветвления. Циклы.

Структурное программирование

- Методология **структурного программирования** – проектирование методом «сверху вниз» (или методом нисходящего проектирования), т.е. написание текста программы от общего к частному, от наиболее общих шагов алгоритма к максимально подробной детализации каждого шага.

- Применение структурного программирования позволяет существенно увеличить скорость написания программ, уменьшить сложность кода, сократить число ошибок и облегчить отладку написанной программы.

- В соответствии с данной методологией любая программа представляет собой набор блоков или модулей, подчиненных общей иерархической схеме.

Законы структурного программирования

- 1. Любая программа представляет собой структуру, построенную из трёх типов базовых конструкций, повторяющих свои аналоги из классических алгоритмов:

a) **последовательное исполнение** – однократное выполнение операций в том порядке, в котором они записаны в тексте программы;

b) **ветвление** – однократное выполнение одной из двух или более операций, в зависимости от выполнения некоторого заданного условия;

c) **цикл** – многократное исполнение одной и той же операции до тех пор, пока выполняется некоторое заданное условие (условие продолжения цикла).

Законы структурного программирования

- 2. В программе базовые конструкции могут быть вложены друг в друга произвольным образом, но никаких других средств управления последовательностью выполнения операций не предусматривается.
- 3. Повторяющиеся фрагменты программы (или же логически завершенные участки кода) могут оформляться в виде подпрограмм (процедур или функций).
- 4. Разработка программы ведётся пошагово, методом нисходящего проектирования.

Плюсы:

- ✓ сокращение логических ошибок;
- ✓ упрощается поиск ошибок;
- ✓ программу можно разрабатывать по частям, независимо друг от друга.

Структурное программирование на Python

Управляющие структуры

- **Управляющие структуры** – операторные конструкции, с помощью которых в программе реализуются **точки ветвления**.

- **Группы управляющих структур:**

  - условные операторы `if, if-else, if-elif-else`;
  - структурное сопоставление с шаблоном* `match-case`;
  - операторы цикла `while, for`;
  - операторы перехода `break, continue`.

** - с версии Python 3.10*

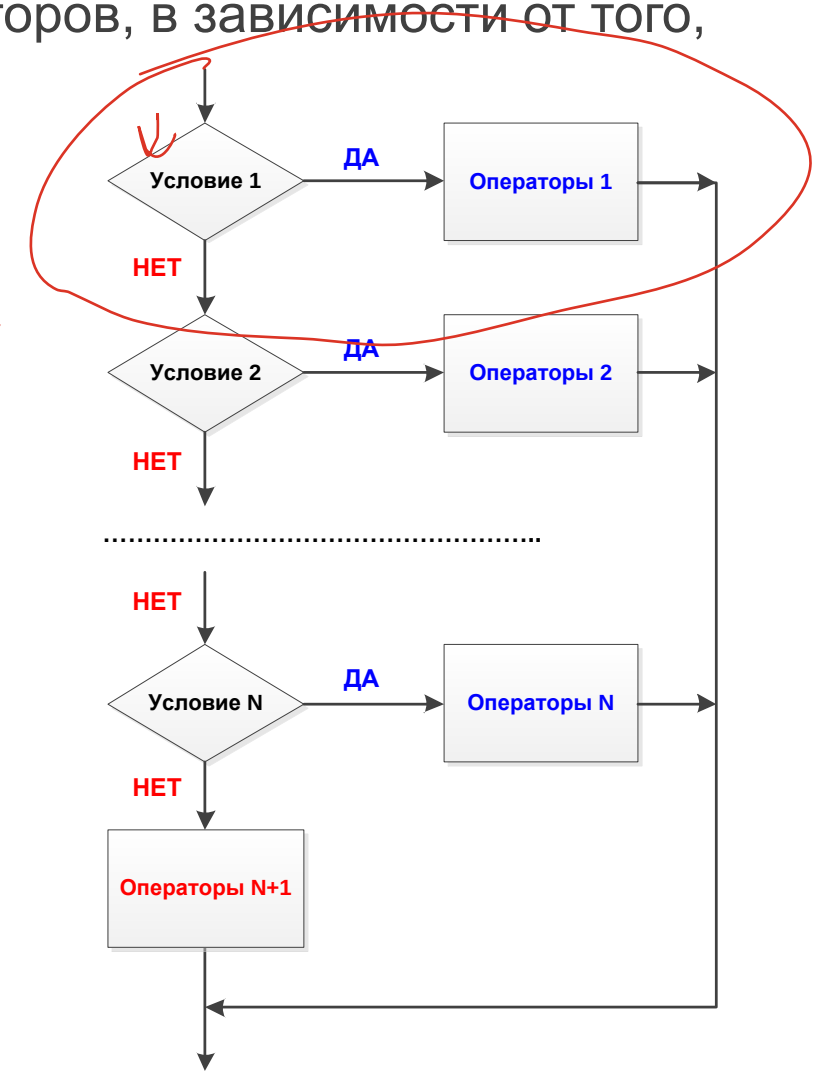
Структурное программирование на Python

Условный оператор ветвления if-elif-else.

- Оператор **if** позволяет выполнять разные блоки операторов, в зависимости от того, выполнятся ли определенное условие.

- Синтаксис вызова:

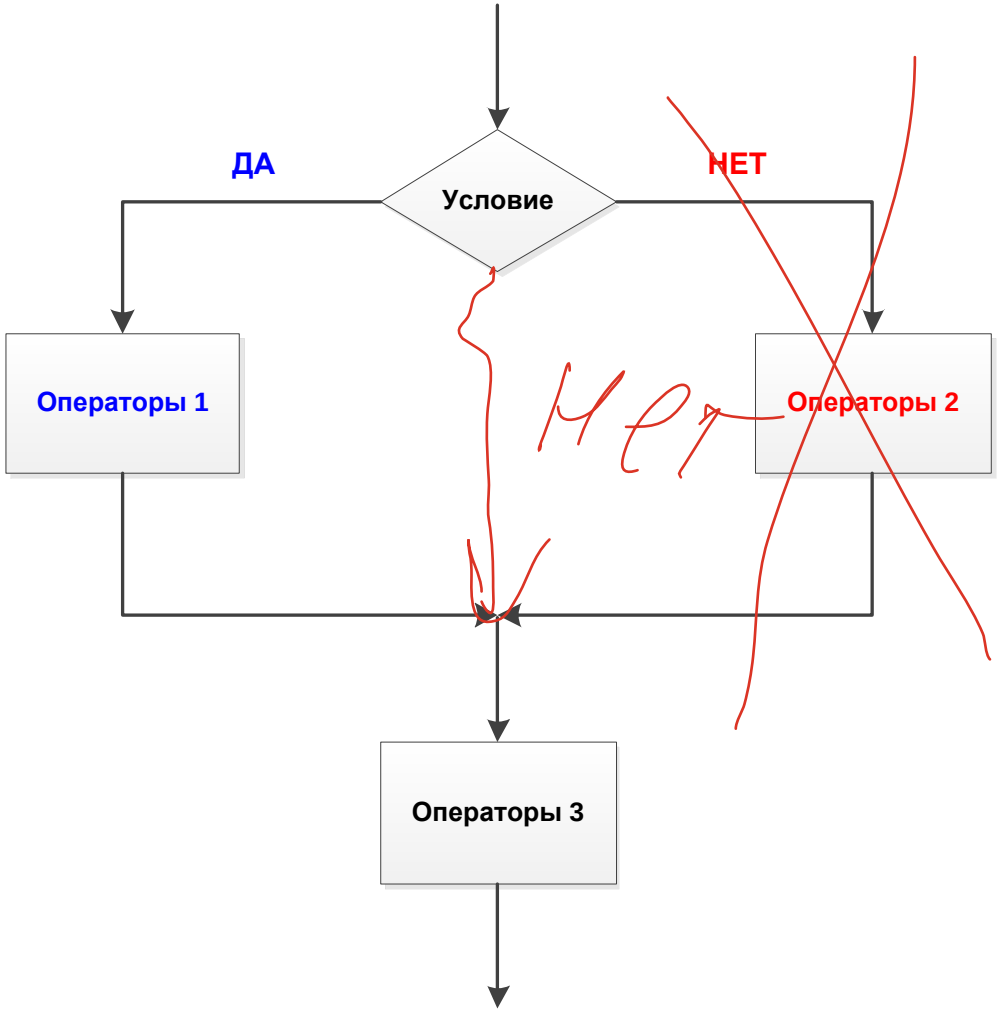
```
if <Логическое выражение> :
    <Блок, выполняемый, если условие истинно>
[elif <Логическое выражение> :
    <Блок, выполняемый, если условие истинно>
]
[elif ...]
[else :
    <Блок, выполняемый, если все условия ложны>
]
```



Структурное программирование на Python

- Блоки внутри составной инструкции выделяются одинаковым количеством пробелов (обычно *четырьмя*).
- Концом блока является инструкция, перед которой расположено меньшее количество пробелов.
- В некоторых языках программирования логическое выражение заключается в круглые скобки. В языке Python это делать необязательно, но можно, т. к. любое выражение может быть расположено внутри круглых скобок.
- Круглые скобки следует использовать при необходимости разместить условие на нескольких строках.

Базовая схема условного оператора



Структурное программирование на Python

- **Пример. Проверка числа на четность.**

- Программа проверяет, является введенное пользователем число четным или нет.
- После проверки выводится соответствующее сообщение.

```
x = int(input("Введите число: "))
if x % 2 == 0:
    print(x, " - чётное число")
else:
    print(x, " - нечётное число")
```

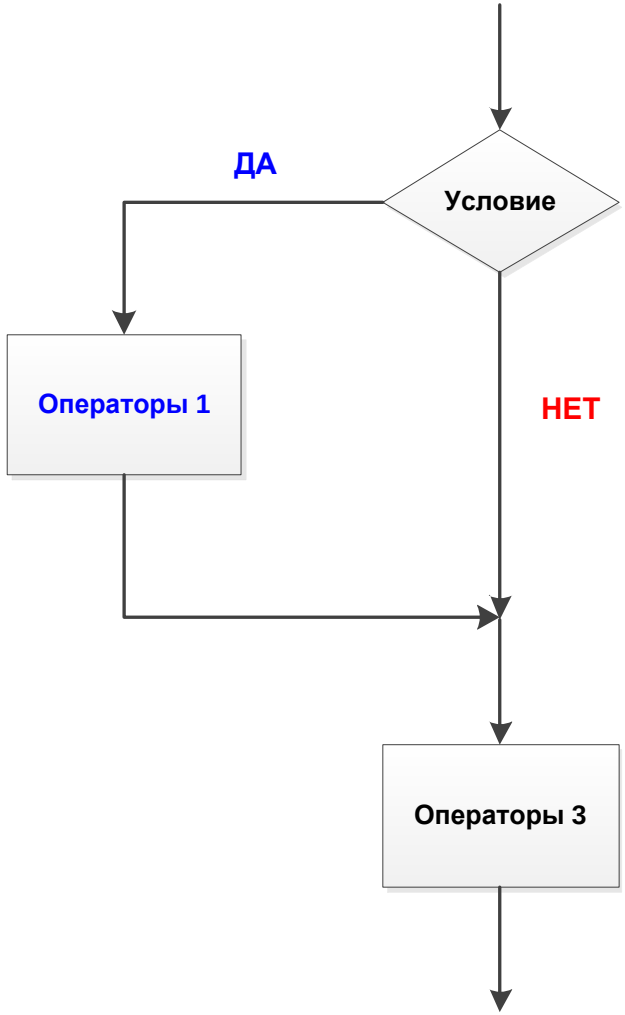
Структурное программирование на Python

Упрощенный вариант условного оператора.

- Синтаксис вызова:

```
if <Логическое выражение> :
    <Блок, выполняемый, если
    условие истинно>
```

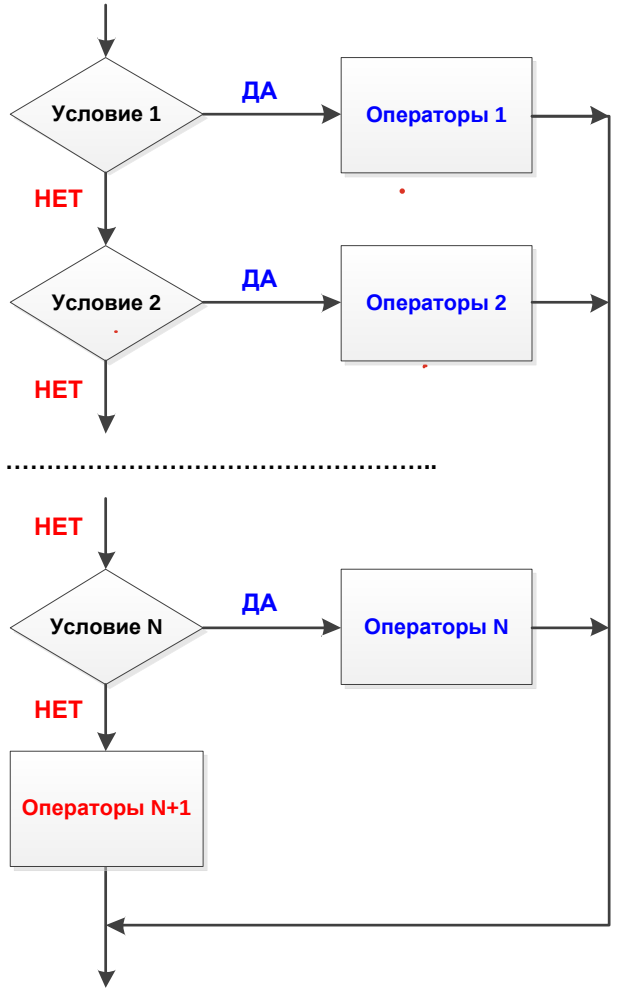
Структурная схема упрощенного варианта условного оператора =>



Структурное программирование на Python

- **Пример. Проверка нескольких условий.**

```
print("""Какой операционной системой вы пользуетесь?
1 – Windows 11
2 – Windows 10
3 – Windows 8
4 – Windows 7
5 – Windows Vista
6 – Другая""")
os = input("Введите число, соответствующее ответу: ")
if os == "1":
    print("Вы выбрали: Windows 11")
elif os == "2":
    print("Вы выбрали: Windows 10")
elif os == "3":
    print("Вы выбрали: Windows 8")
elif os == "4":
    print("Вы выбрали: Windows 7")
elif os == "5":
    print("Вы выбрали: Windows Vista")
elif os == "6":
    print("Вы выбрали: другая")
elif not os:
    print("Вы не ввели число")
else:
    print("Мы не смогли определить вашу операционную систему")
```



Структурное программирование на Python

- **Пример. Вложенные инструкции.**

```
print("""Какой операционной системой вы пользуетесь?
1 – Windows 11
2 – Windows 10
3 – Windows 8
4 – Windows 7
5 – Windows Vista
6 – Другая""")
os = input("Введите число, соответствующее ответу: ")
if os != "":
    if   os == "1":
        print("Вы выбрали: Windows 11")
    elif os == "2":
        print("Вы выбрали: Windows 10")
    elif os == "3":
        print("Вы выбрали: Windows 8")
    elif os == "4":
        print("Вы выбрали: Windows 7")
    elif os == "5":
        print("Вы выбрали: Windows Vista")
    elif os == "6":
        print("Вы выбрали: другая")
    else:
        print("Мы не смогли определить вашу операционную систему")
else:
    print("Вы не ввели число")
```

Структурное программирование на Python

- match-case - структурное сопоставление с шаблоном (PEP 635, начиная с Python 3.10.0)

```
print("""Какой операционной системой вы пользуетесь?
1 — Windows 11
2 — Windows 10
3 — Windows 8
4 — Windows 7
5 — Windows Vista
6 — Другая""")
os = input("Введите число, соответствующее ответу: ")
match os:
    case "1": print("Вы выбрали: Windows 11")
    case "2": print("Вы выбрали: Windows 10")
    case "3": print("Вы выбрали: Windows 8")
    case "4": print("Вы выбрали: Windows 7")
    case "5": print("Вы выбрали: Windows Vista")
    case "6": print("Вы выбрали: другая")
    case "":  print("Вы не ввели число")
    case _: # Во всех остальных случаях
        print("Мы не смогли определить вашу операционную систему")
input("\nДля завершения работы нажмите клавишу Enter...")
```

Циклы в языке Python

- ***Операторы цикла позволяют многократно выполнять серии однотипных действий, которые выполняются до тех пор пока остается справедливым (или пока не будет выполнено) определенное условие.***

Циклы в языке Python

Цикл while.

Выполнение инструкций в цикле **while** продолжается до тех пор, пока логическое выражение истинно.

Цикл **while** имеет следующий формат:

```
<Задание начального значения для переменной-счетчика>
while <Условие>:
    <Инструкции>
    <Приращение значения в переменной-счетчике>
[ else:
    <Блок, выполняемый, если не использовался оператор break>
]
```

Циклы в языке Python

Цикл while.

Последовательность работы цикла while :

1. Переменной-счетчику присваивается начальное значение.

2. Проверяется условие, и если оно истинно, то выполняются инструкции внутри цикла, иначе выполнение цикла завершается.

3. Переменная-счетчик изменяется на величину, указанную в параметре <приращение>.

4. Переход к пункту 2.

5. Если внутри цикла не использовался оператор break, то после завершения выполнения цикла будет выполнен блок в инструкции else. Этот блок не является обязательным.

Циклы в языке Python

Цикл while.

- **Пример. Вывод чисел от 1 до 100**

```
i = 1                   # Начальное значение
while i < 101:          # Условие
    print(i)            # Инструкция
    i += 1              # Приращение
```

- **Пример. Вывод чисел от 100 до 1**

```
i = 100
while i!=0:
    print(i)
    i -= 1
```

Циклы в языке Python

Оператор continue.

- Оператор **continue** позволяет перейти к следующей итерации цикла до завершения выполнения всех инструкций внутри цикла.

- **Пример.** Выведем все числа от 1 до 100, кроме чисел от 5 до 10 включительно.

```
i = 0
while i < 100:
    i += 1
    if 4 < i < 11:
        continue # На следующую итерацию
    print(i)
```

Циклы в языке Python

Оператор break.

- Оператор **break** позволяет прервать выполнение цикла досрочно.
- **Пример.** Выведем все числа от 1 до 100 ещё одним способом.

```
i = 1
while True:
    if i > 100: break   # Прерываем цикл
    print(i)
    i += 1
```

- В условии указано значение **True.** В этом случае выражения внутри цикла станут выполняться ***бесконечно***. Использование оператора **break** прерывает выполнение цикла, как только он будет выполнен 100 раз.
- *Внимание!*

  Оператор **break** прерывает выполнение цикла, а не программы, т. е. далее будет выполнена инструкция, следующая сразу за циклом.

Циклы в языке Python

Цикл перебора for.

- Цикл for применяется для перебора элементов последовательности.

- Формат цикла for:

```
for <Текущий элемент> in <Последовательность>:
    <Инструкции внутри цикла>
[else:
    <Блок, выполняемый, если не использовался оператор break>
]
```

Циклы в языке Python

Конструкции в цикле for.

- **<Последовательность>** — объект, поддерживающий механизм итерации: строка, список, кортеж, диапазон и др.;
- **<Текущий элемент>** — на каждой итерации через эту переменную доступен очередной элемент последовательности или ключ словаря;
- **<Инструкции внутри цикла>** — блок, который будет многократно выполняться;
- Если внутри цикла не использовался оператор **break**, то после завершения выполнения цикла будет выполнен блок в инструкции **else**. Этот блок не является обязательным.

Циклы в языке Python

Цикл for. Примеры.

- **Перебор букв в слове.**

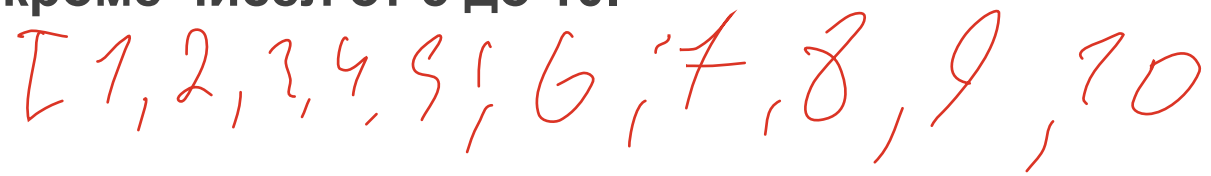
```
for s in "str":
    print(s, end=" ") # Печать букв через пробел
```

- **Перебор списка и кортежа.**

```
for x in [1, 2, 3]:
    print(x)
for y in (1, 2, 3):
    print(y)
```

- **Вывод всех чисел от 1 до 100, кроме чисел от 5 до 10.**

```
for i in range(1, 101):
    if 4 < i < 11:
        continue    # На следующую итерацию
    print(i)
```



Циклы в языке Python

Цикл по итератору. Функция enumerate().

- Функция `enumerate(<Объект>[, start=0])` на каждой итерации цикла `for` возвращает кортеж из индекса и значения текущего элемента.
- С помощью необязательного параметра **start** можно задать начальное значение индекса.

- Пример.
- Умножить на 2 каждый элемент списка, который в качестве значения содержит четное число.

```
# Цикл по итератору
arr = [11, 22, 55, 66, 77, 88]
for i, elem in enumerate(arr):
    if elem % 2 == 0:
        arr[i] *= 2
print(arr)
# Результат выполнения: [11, 44, 55, 132, 77, 176]
```

ЦЕНТР
ДОПОЛНИТЕЛЬНОГО
ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

do.bmstu.ru