Условные конструкции

№ урока: 3 **Курс:** Python Starter

Средства обучения: PyCharm

Обзор, цель и назначение урока

В уроке рассматриваются условные операторы, при помощи которых можно реализовывать алгоритмы с ветвлениями.

Изучив материал данного занятия, учащийся сможет:

- Составлять алгоритмы с ветвлениями
- Пользоваться условными операторами в Python

Содержание урока

1. Понятие условных конструкций
2. Оператор if-else
3. Оператор if-elif-else
4. if в виде выражения
5. Логические значения не-булевских выражений

Резюме

Условные конструкции
Вам нужно научить робота делать бутерброд со сгущёнкой. Ваши инструкции могут быть примерно такими:

- Открыть нижнюю правую дверцу шкафчика.
- Взять банку со сгущёнкой и вынуть ее из шкафчика.
- Закрыть шкафчик.
- Держа банку со сгущёнкой в левой руке, взять крышку в правую руку.
- Поворачивать правую руку против часовой стрелки, пока крышка не откроется.
- И так далее …

Это программа: вы описали каждый шаг, который компьютер должен выполнить, и указали всю информацию, которая требуется компьютеру для выполнения каждого шага. А теперь представьте, что вы объясняете человеку, как сделать бутерброд со сгущёнкой. Ваши инструкции будут, скорее всего, такими:

- Достаньте сгущёнку и хлеб.
- Намажьте ножом или ложкой сгущёнку на один ломтик хлеба.
- Приятного аппетита!

Это алгоритм: процесс, которому нужно следовать для получения желаемого результата (в данном случае бутерброда со сгущёнкой). Обратите внимание, что **алгоритм более обобщённый, чем программа**.
Программа сообщает роботу, откуда именно нужно взять предметы на конкретной кухне, с точным указанием всех необходимых деталей. Это **реализация** алгоритма, которая предоставляет все важные детали, но может быть выполнена на любом оборудовании (в данном случае - на кухне), со всеми необходимыми элементами.
В итоге, **алгоритм** — набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное число действий.
Раньше вместо слова «порядок» использовалось слово «последовательность». По мере развития параллельности в работе компьютеров слово «последовательность» стали заменять более общим словом «порядок». Это связано с тем, что работа каких-то инструкций алгоритма может быть зависима от других инструкций или результатов их работы.

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Таким образом, некоторые инструкции должны выполняться строго после завершения работы инструкций, от которых они зависят. Независимые инструкции или инструкции, ставшие независимыми из-за завершения работы инструкций, от которых они зависят, могут выполняться в произвольном порядке, параллельно или одновременно, если это позволяют используемые процессор и операционная система.
Часто в качестве исполнителя выступает некоторый механизм (компьютер, токарный станок, швейная машина), но понятие алгоритма необязательно относится к компьютерным программам. Например, чётко описанный рецепт приготовления блюда также является алгоритмом, в таком случае исполнителем является человек.
Формальные свойства алгоритмов:

- **Детерминированность** (определённость). В каждый момент времени следующий шаг работы однозначно определяется состоянием системы. Таким образом, алгоритм выдаёт один и тот же результат (ответ) для одних и тех же исходных данных.
- **Понятность** — алгоритм должен включать только те команды, которые доступны исполнителю и входят в его систему команд.
- **Конечность** — при корректно заданных исходных данных алгоритм должен завершать работу и выдавать результат за конечное число шагов.
- **Массовость** (универсальность). Алгоритм должен быть применим к разным наборам исходных данных.
- **Результативность** — завершение алгоритма определёнными результатами.

Алгоритм содержит ошибки, если приводит к получению неправильных результатов либо не даёт результатов вовсе. Алгоритм не содержит ошибок, если он даёт правильные результаты для любых допустимых исходных данных.
Алгоритмы бывают:

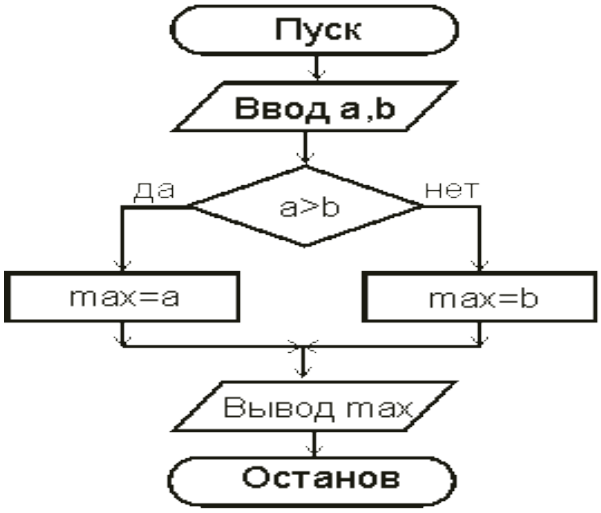
- **Линейные** — набор команд (указаний), выполняемых последовательно во времени друг за другом.
- **Разветвляющийся** — алгоритм, содержащий хотя бы одно условие, в результате проверки которого может осуществляться разделение на несколько параллельных ветвей алгоритма.
- **Циклические** — алгоритм, предусматривающий многократное повторение одного и того же действия (одних и тех же операций) над новыми исходными данными. К циклическим алгоритмам сводится большинство методов вычислений, перебора вариантов.
- **Смешанные** - объединение нескольких видов алгоритмов.

Цикл программы — последовательность команд (тело цикла), которая может выполняться многократно до удовлетворения некоторого условия.
Линейные алгоритмы состоят из последовательного набора команд, которые необходимо выполнить одна за другой. Программы, реализующие именно такие алгоритмы, создавались на предыдущем уроке.
Пример линейного алгоритма:

Пример линейного алгоритма

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Алгоритм с ветвлениями содержит условия, в зависимости от которых выполняется одна или другая последовательность действий. Каждая из них, в свою очередь, также может (но не обязана) содержать ветвления.
Пример алгоритма с ветвлением:



Пример алгоритма с ветвлением

Оператор if (if-else)

В Python для реализации ветвлений используется оператор if. Самой простой его формой является:

```
if условие:
    операторы
```

где условие – это логическое выражение (*см. урок 2*), а операторы – это последовательность каких-либо других команд.

Пример:

```
x = 7

if x > 5:
    print('x больше пяти')
```

Блок операторов не может быть пустым. Если необходимо создать блок, который ничего не делает (например, чтобы набросать структуру кода, а затем реализовывать отдельные его части, или в каких-нибудь других целях), то используется специальный оператор pass.

Оператор pass не выполняет никаких действий и обычно используется, когда нужно создать логически пустой блок кода там, где синтаксис языка требует наличия каких-либо операторов.
Пример:

```
value = None

if value is None:
    pass  # TO-DO: исправить позже
```

Очень важны в коде на Python отступы, так как именно по ним интерпретатор определяет границы блоков кода и уровни их вложенности. Операторы, находящиеся внутри одного блока, должны выделятся слева одинаковым количеством пробелов или знаков табуляции. Хорошим стилем является внутри одного файла выделять каждый блок одинаковым образом, символами пробела или табуляции в количестве, кратным какому-либо определённому числу, и очень нежелательно смешивать пробелы и знаки табуляции.

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Согласно соглашениям написания кода, на Python (*PEP 8*), принято выделять каждый уровень вложенности четырьмя пробелами. При написании кода в интегрированной среде разработки, такой как PyCharm или Visual Studio с расширением Python Tools for Visual Studio, об этом правиле можно не задумываться, так как редактор кода гарантирует его автоматическую поддержку и правильное форматирование, однако при редактировании кода в обычном редакторе (если возникает такая необходимость), нужно понимать, как именно расставляются отступы и проверять правильность самостоятельно.
Оператор if также можно использовать в однострочной форме:

if условие: операторы

Если блок операторов здесь состоит больше, чем из одной инструкции, то они разделяются точкой с запятой («;»). Однако такая форма нежелательна для использования, так как значительно снижает читабельность кода, и допустима только если блок операторов состоит только из одной инструкции, и как она, так и условие, достаточно короткие. Но даже в этом случае предпочтительнее всё же размещать её на новой строке.
Рассмотренная выше форма оператора if выполняет или не выполняет ветку кода в зависимости от истинности заданного выражения. Однако, как было сказано выше, разветвляющиеся алгоритмы характерны тем, что в зависимости от значения логического выражения может быть выполнена одна из двух веток кода. Такая конструкция в Python реализуется оператором if-else:

```
if условие:
    блок_операторов_1
else:
    блок_операторов_2
```

Если условие истинно, то выполняются операторы из первого блока, иначе – второго.

В качестве примера рассмотрим реализацию алгоритма из блок-схемы выше:

```
x = int(input('x = '))
if x > 0:
    y = x ** 0.5
else:
    y = x ** 2
print(y)
```

Так как блоки операторов внутри if могут содержать любые операторы и их последовательности, очевидно, что операторы if могут быть вложенными. Пример:

```
x = int(input('x = '))

if 0 < x < 7:
    print('Значение x входит в заданный диапазон, продолжаем')
    y = 2 * x - 5
    if y < 0:
        print('Значение y отрицательно')
    else:
        if y > 0:
            print('Значение y положительно')
        else:
            print('y = 0')
```

Оператор if-elif-else

В последнем примере, во втором вложенном if мы видим ситуацию, в которой мы поочерёдно проверяем несколько условий (в данном случае, отрицателен ли y, положителен или равен нулю). В общем случае это можно записать так:

```
if условие1:
    команды1
else:
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
if условие2:
    командыЗ
else:
    if условие3:
        командыЗ
      # ...
    else:
        if условиеN:
            командыN
        else:
            командыM
```

Подобный приём называется каскадированием условных операторов. Ситуации, в которых необходимо выполнить подобную проверку, встречаются очень часто, однако видно, что каскад условных операторов выглядит несколько нечитабельно и легко запутаться при его написании. В разных языках программирования есть разные подходы к решению этой проблемы.

Для языков, унаследовавших синтаксис от C (C, C++, Java, C# и т.д.) характерен оператор переключения switch (в не C-подобных языках, которые также имеют этот оператор, он может называться case или ещё как-нибудь иначе). В зависимости от значения выражения он выполняет одну из веток кода. Его недостатком является то, что в большинстве языков, где он реализован (исключения – C#, JavaScript) управляющее выражение может иметь только целый тип, а условия переключения могут проверять его значение только на равенство определённым константам. В большинстве случаев этого достаточно, но в других приходится использовать каскад операторов if.
В функциональных языках программирования, таких как Haskell или F#, есть довольно мощный механизм сопоставления с образцом.
В Python используется крайне простое и, тем не менее, гибкое и удобное решение этой проблемы: оператор ветвления с несколькими условиями.

```
if условие1:
    операторы1
elif условие2:
    операторы2
elif условие3:
    операторы3
# ...
else:
    операторыN
```

(как и в обычной форме оператора if, ветка else здесь необязательна).

Таким образом, в общем виде оператор if в Python выглядит так: сначала идёт ключевое слово if, условие, двоеточие и блок кода, затем может идти любое количество необязательных блоков elif, в конце может указываться необязательных блок else.
Сначала интерпретатор проверяет истинность условия в блоке if, и если оно истинно, то выполняет соответствующие команды и переходит к коду после всего оператора if, иначе, если присутствуют блоки elif, проверяет истинность соответствующих условий в каждом блоке по очереди, пока не найдёт истинное, после чего выполняет соответствующие команды и завершает выполнение оператора if. Если же все условия в блоках if и elif были ложными, то при наличии секции else, выполняются операторы в ней.

if в виде выражения (условное выражение)

Довольно часто возникает ситуация, когда нам нужно использовать то или иное выражение в зависимости от определённого условия. Поэтому существует такая конструкция, как условное выражение (или, иначе, «**тернарный оператор**»).

Синтаксис условного выражения в Python:
выражение1 if условие else выражение2

CyberBionic systematics
CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Результатом всего этого выражения является значение первого выражения, если условие истинно, или второго, если ложно.
Пример. Вместо

```
is_ready = input("Are you ready?: ")

if is_ready:
    state_msg = 'Ready'
else:
    state_msg = 'Not ready yet'
```

можно записать

```
is_ready = input("Are you ready?: ")

x = 'Ready' if is_ready else 'Not ready yet'
```

Логические значения не-булевских выражений

Любой объект в Python может быть рассмотрен как логическое значение для использования в условии оператора if, конструирования значения типа bool или использования как операнда логических операций.

Изученные в предыдущем уроке типы (NoneType, bool, int, float, complex, str) отображаются на логические значения так: None, False, 0, 0.0, 0j и '' (пустая строка) считаются ложными значениями, все остальные – истинными.
Пример. Вместо

```
string = input("Type some string: ")

if string is not None and string != '':
    num = int(string)
```

Можно записать:

```
string = input("Type some string: ")

if string:
    num = int(string)
```

Закрепление материала

- Что такое алгоритм?
- Что такое разветвляющийся алгоритм?
- Как создавать ветвления в программах на Python?
- Как выглядит оператор if в Python? Охарактеризуйте его основные секции.
- Зачем нужен оператор pass?
- Зачем нужен оператор ветвления с несколькими условиями?
- Что такое условное выражение?
- Каким образом значения не-булевских типов могут быть рассмотрены как логические значения и зачем?

Дополнительное задание

Задание
Напишите программу-калькулятор, в которой пользователь сможет ввести выбрать операцию, ввести необходимые числа и получить результат. Операции, которые необходимо реализовать: сложение, вычитание, умножение, деление, возведение в степень, синус, косинус и тангенс числа.

Самостоятельная деятельность учащегося

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Задание 1
Напишите программу, которая спрашивает у пользователя его имя, и если оно совпадает с вашим, выдаёт определённое сообщение.

Задание 2
Напишите программу, которая вычисляет значение функции

$$y = \begin{cases} \cos(3x), & -\pi \le x \le \pi \\ x, & x < -\pi \text{ или } x > \pi \end{cases}$$

Задание 3
Напишите программу, которая решает квадратное уравнение $ax^2 + bx + c = 0$ в действительных числах. В отличие от аналогичного упражнения из прошлого урока, программа должна выдавать сообщение об отсутствии действительных корней, если значение дискриминанта $D = b^2 - 4ac$ отрицательно, единственное решение $x = -\frac{b}{2a}$, если он равен нулю, или два корня $x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$, если он положителен.

Рекомендуемые ресурсы

Документация по Python
https://docs.python.org/3/tutorial/controlflow.html#if-statements
https://docs.python.org/3/reference/compound_stmts.html#the-if-statement
https://docs.python.org/3/reference/expressions.html#conditional-expressions
https://docs.python.org/3/library/stdtypes.html#truth-value-testing

Статьи в Википедии о ключевых понятиях, рассмотренных на этом уроке
https://ru.wikipedia.org/wiki/Алгоритм
https://ru.wikipedia.org/wiki/Оператор_ветвления

CyberBionic systematics
CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com