# Умовні конструкції

**№ уроку:** 3 **Курс:** Python Starter

**Засоби навчання:** PyCharm

## Огляд, мета та призначення уроку

Після завершення уроку учні матимуть уявлення про умовні оператори, за допомогою яких можна реалізовувати алгоритми з розгалуженнями, умовні вирази (тернарні оператори), а також конструкції match/case, котрі реалізовані у Python версії 3.10.

## Вивчивши матеріал даного заняття, учень зможе:

- Складати алгоритми з розгалуженнями
- Користуватися умовними операторами в Python
- Оптимізувати реалізацію умовних алгоритмів

## Зміст уроку

1. Поняття умовних конструкцій
2. Оператор if-else
3. Оператор if-elif-else
4. Тернарні оператори
5. Логічні значення не-булевих виразів
6. Конструкція match/case

## Резюме

### Умовні конструкції

Вам потрібно навчити робота робити бутерброд зі згущеним молоком. Ваші інструкції можуть бути приблизно такими:

- Відкрити нижні праві дверцята шафи.
- Взяти банку зі згущеним молоком і вийняти її з шафи.
- Закрити шафу.
- Тримаючи банку зі згущеним молоком у лівій руці, взяти кришку в праву руку.
- Повертати праву руку проти годинникової стрілки, доки кришка не відкриється.
- І так далі …

Це програма: ви описали кожен крок, який комп'ютер повинен виконати, і вказали всю інформацію, яка потрібна комп'ютеру для виконання кожного кроку. А тепер уявіть, що ви пояснюєте людині, як зробити бутерброд зі згущеним молоком. Ваші інструкції будуть, швидше за все, такими:

- Дістаньте згущене молоко і хліб.
- Намажте ножем або ложкою згущене молоко на одну скибочку хліба.
- Смачного!

Це алгоритм: процес, якому потрібно слідувати для отримання бажаного результату (в даному випадку бутерброду зі згущеним молоком). Зверніть увагу, що **алгоритм більш узагальнений, ніж програма**.
Програма повідомляє роботу, звідки саме потрібно взяти предмети на конкретній кухні з точною вказівкою всіх необхідних деталей. Це реалізація алгоритму, яка надає всі важливі деталі, але може бути виконана на будь-якому устаткуванні (в даному випадку – на кухні), з усіма необхідними елементами.
У результаті, **алгоритм** — набір інструкцій, що описують порядок дій виконавця для досягнення результату рішення задачі за кінцеве число дій.



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

*Раніше замість слова "порядок" використовувалося слово "послідовність". Із розвитком паралельності у роботі комп'ютерів слово «послідовність» стали замінювати більш загальним словом «порядок». Це пов'язано з тим, що робота якихось інструкцій алгоритму може бути залежна від інших інструкцій чи результатів їх роботи.*

Таким чином, деякі інструкції повинні виконуватися лише після завершення роботи інструкцій, від яких вони залежать. Незалежні інструкції або інструкції, які стали незалежними через завершення роботи інструкцій, від яких вони залежать, можуть виконуватися в довільному порядку, паралельно або одночасно, якщо це дозволяють процесор і операційна система, що використовуються.

Часто виконавцем виступає певний механізм (комп'ютер, токарний верстат, швейна машина), але поняття алгоритму необов'язково відноситься до комп'ютерних програм. Наприклад, чітко описаний рецепт приготування страви є алгоритмом, у такому випадку виконавцем є людина.

Формальні властивості алгоритмів:

- **Дискретність** — алгоритм повинен представляти процес розв'язання задачі як послідовне виконання деяких простих кроків. При цьому для виконання кожного кроку алгоритму потрібно кінцевий відрізок часу, тобто перетворення вихідних даних у результат здійснюється дискретно в часі.
- **Детермінованість (визначеність)**. У кожен момент часу наступний крок роботи однозначно визначається станом системи. Таким чином, алгоритм видає один і той самий результат (відповідь) для одних і тих самих вихідних даних.
- **Зрозумілість** — алгоритм повинен включати лише ті команди, що доступні виконавцю і входять у його систему команд.
- **Кінцевість (завершуваність)** — при коректно заданих вихідних даних алгоритм повинен завершувати роботу і видавати результат за кінцеве число кроків.
- **Масовість** (універсальність). Алгоритм має бути застосовуваний до різних наборів вихідних даних.
- **Результативність** — завершення алгоритму певними результатами.

Алгоритм містить помилки, якщо призводить до отримання неправильних результатів або не дає результатів зовсім. Алгоритм не містить помилок, якщо він дає правильні результати для будь-яких допустимих вихідних даних.

Алгоритми бувають:

- **Лінійні** — набір команд (вказівок), які виконуються послідовно в часі один за одним.
- **Розгалужувані** — алгоритм, що містить хоча б одну умову, в результаті перевірки якої може здійснюватися поділ на кілька паралельних гілок алгоритму.
- **Циклічні** — алгоритм, що передбачає багаторазове повторення однієї й тієї ж дії (одних і тих самих операцій) над новими вихідними даними. До циклічних алгоритмів зводиться більшість методів обчислень, перебору варіантів.
- **Змішані** – поєднання кількох видів алгоритмів.

**Цикл програми** – послідовність команд (тіло циклу), яка може виконуватися багаторазово до задоволення певної умови.

Лінійні алгоритми складаються із послідовного набору команд, які необхідно виконати одна за одною. Програми, що реалізують такі алгоритми, створювалися на попередньому уроці.

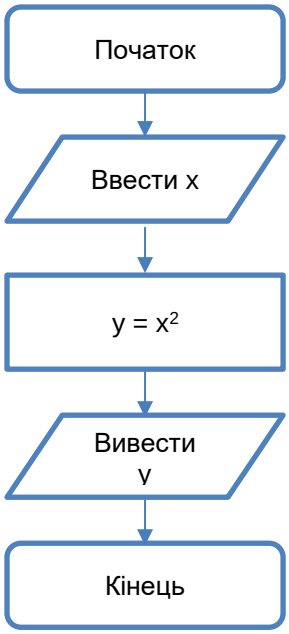
Приклад лінійного алгоритму:



CyberBionic systematics

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

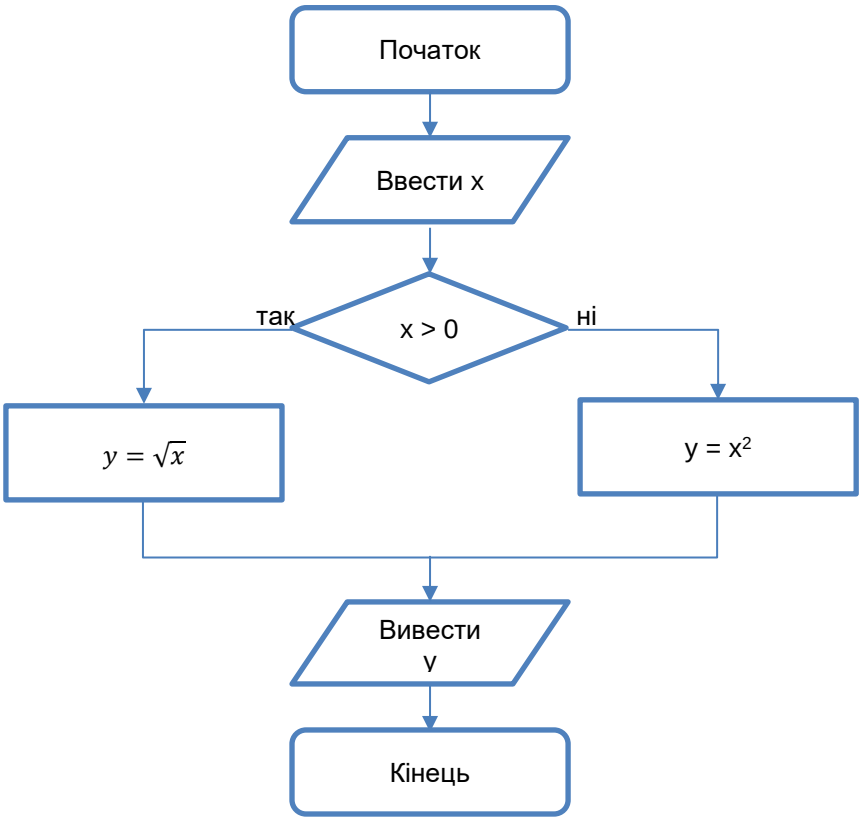
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com




*Приклад лінійного алгоритму*

Алгоритм із розгалуженнями містить умови, в залежності від яких виконується одна чи інша послідовність дій. Кожна з них, у свою чергу, також може (але не зобов'язана) мати розгалуження.

Приклад алгоритму з розгалуженням:




*Приклад алгоритму з розгалуженням*



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

## Оператор if (if-else)

У Python для реалізації розгалужень використовується оператор if. Найпростішою його формою є:

```
if умова:
    оператори
```

де умова – це логічний вираз (див. урок 2), а оператори – це послідовність будь-яких інших команд.

Приклад:

```
x = 7

if x > 5:
    print('x більше п'яти')
```

Блок операторів не може бути порожнім. Якщо необхідно створити блок, який нічого не робить (наприклад, щоб накидати структуру коду, а потім реалізовувати окремі його частини, або з будь-якою іншою метою), то використовується спеціальний оператор pass.

Оператор pass не виконує жодних дій і зазвичай використовується, коли потрібно створити логічно порожній блок коду там, де синтаксис мови потребує будь-яких операторів.
Приклад:

```
value = None

if value is None:
    pass  # TO-DO: виправити пізніше
```

Дуже важливі в коді на Python відступи, оскільки саме за ними інтерпретатор визначає межі блоків коду та рівні їх вкладеності. Оператори, що знаходяться всередині одного блоку, повинні виділятися зліва однаковою кількістю пробілів або знаків табуляції. Хорошим стилем є всередині одного файлу виділяти кожен блок однаковим чином, символами пробілу або табуляції в кількості, кратної певному числу, і дуже небажано змішувати пробіли і знаки табуляції.

Відповідно до угод написання коду, на Python (PEP 8) прийнято виділяти кожен рівень вкладеності чотирма пробілами. При написанні коду в інтегрованому середовищі розробки, такому як PyCharm або Visual Studio з розширенням Python Tools for Visual Studio, про це правило можна не замислюватися, оскільки редактор коду гарантує його автоматичну підтримку та правильне форматування, проте при редагуванні коду у звичайному редакторі (якщо виникає така необхідність), треба розуміти, як саме розставляються відступи та перевіряти правильність самостійно.
Оператор if також можна використовувати в однорядковій формі:

```
if умова: оператори
```

Якщо блок операторів тут складається більше, ніж з однієї інструкції, вони розділяються точкою з комою («;»). Однак така форма небажана для використання, оскільки значно знижує читабельність коду, і допустима тільки якщо блок операторів складається тільки з однієї інструкції, і як вона, так і умова, досить короткі. Але навіть у цьому випадку краще все ж таки розміщувати її на новому рядку.
Розглянута вище форма оператора if виконує чи не виконує гілку коду залежно від істинності заданого виразу. Однак, як було сказано вище, алгоритми, що розгалужуються, характерні тим, що в залежності від значення логічного виразу може бути виконана одна з двох гілок коду. Така конструкція в Python реалізується оператором if-else:

```
if умова:
    блок_операторів_1
```



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

```
else:
   блок_операторів_2
```

Якщо умова істинна, то виконуються оператори з першого блоку, інакше – з другого.

Як приклад розглянемо реалізацію алгоритму із блок-схеми вище:

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

Оскільки блоки операторів усередині if можуть містити будь-які оператори та їх послідовності, очевидно, що оператори if можуть бути вкладеними. Приклад:

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

if 0 < x < 7:
   print('Значення x входить до заданого діапазону, продовжуємо')
   y = 2 * x - 5
   if y < 0:
      print('Значення y від'ємне)
   else:
      if y > 0:
         print('Значення y додатнє)
      else:
         print('y = 0')
```

### Оператор if-elif-else

В останньому прикладі, в другому вкладеному if ми бачимо ситуацію, в якій ми по черзі перевіряємо кілька умов (у даному випадку, чи негативний y, позитивний або дорівнює нулю). Загалом це можна записати так:

```
if умова1:
   команди1
else:
   if умова2:
      команди2
   else:
      if умова3:
         команди3
       # ...
      else:
         if умоваN:
            командиN
         else:
            командиM
```

Подібний прийом називається каскадуванням умовних операторів. Ситуації, в яких необхідно виконати подібну перевірку, трапляються дуже часто, проте видно, що каскад умовних операторів виглядає дещо нечитабельним і легко заплутатися під час його написання. У різних мовах програмування є різні підходи до вирішення цієї проблеми.

Для мов, які успадкували синтаксис від C (C, C++, Java, C# і т. д.) характерний оператор перемикання switch (у не C-подібних мовах, які також мають цей оператор, він може називатися case або ще якось інакше). Залежно від значення виразу він виконує одну із гілок коду. Його недоліком є те, що у більшості мов, де він реалізований (виключення – C#, JavaScript) керуючий вираз може мати лише цілий тип, а умови перемикання можуть перевіряти його значення лише



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

на рівність певним константам. Найчастіше цього достатньо, але в інших випадках доводиться використовувати каскад операторів if.
У функціональних мовах програмування, таких як Haskell або F# є досить потужний механізм співставлення зі зразком.
У Python використовується вкрай просте і, тим не менш, гнучке та зручне вирішення цієї проблеми: оператор розгалуження з кількома умовами.

```
if умова1:
   оператори1
elif умова2:
   оператори2
elif умова3:
   оператори3
# ...
else:
   операторыN
```

(як і у звичайній формі оператора if, гілка else тут необов'язкова).

Таким чином, в загальному вигляді оператор if в Python виглядає так: спочатку йде ключове слово if, умова, двокрапка та блок коду, потім може йти будь-яка кількість необов'язкових блоків elif, в кінці може вказуватися необов'язковий блок else.
Спочатку інтерпретатор перевіряє істинність умови в блоці if, і якщо воно істинно, то виконує відповідні команди і переходить до коду після всього оператора if, інакше, якщо є блоки elif, перевіряє істинність відповідних умов у кожному блоці по черзі, доки не знайде істинне, після чого виконує відповідні команди та завершує виконання оператора if. Якщо ж усі умови в блоках if та elif були хибними, то за наявності секції else виконуються оператори у ній.

Тернарні оператори

Досить часто виникає ситуація, коли нам потрібно використовувати той чи інший вираз залежно від певної умови. Тому існує така конструкція, як умовний вираз (або інакше **«тернарний оператор»**).

Синтаксис умовного виразу в Python:
вираз1 if умова else вираз2

Результатом всього цього виразу є значення першого виразу, якщо умова істинна, чи другого, якщо хибна.

Приклад 1. Замість:

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

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

можна записати:

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

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

Приклад 2: # Проста умова

```
number = int(input("Input your digit: "))

if number >= 0:
```



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

```
    print(number)
else:
    print(-number)
```

Еквівалентно:

```
# Тернарний оператор
print(number if number >= 0 else -number)
```

Приклад 3:

```
value_str = "hello"
result = None if not value_str else value_str
print(result)
another_way = value_str if value_str else None
print(another_way)
```

### Логічні значення не-булевих виразів

Будь-який об'єкт у Python може бути розглянутий як логічне значення для використання в умові оператора if, конструювання значення типу bool або використання як операнда логічних операцій.

Вивчені в попередньому уроці типи (NoneType, bool, int, float, complex, str) відображаються на логічні значення так: None, False, 0, 0.0, 0j та '' (порожній рядок) вважаються хибними значеннями, решта – істинними.

Приклад. Замість

```
string = input("Type some string: ")
if string is not None and string != '':
    num = int(string)
```

Можна записати:

```
string = input("Type some string: ")
if string:
    num = int(string)
```

**Конструкція match/case**

Нещодавно (а саме – 04.10.2021 року) офіційно побачила світ ювілейна версія мови Python, версія 3.10. У ній було додано кілька змін, а найцікавішим було введення pattern matching statement (оператор співставлення зі зразком). Як повідомляє офіційний опис цього оператора в PEP622, розробники більшою мірою надихалися напрацюваннями таких мов як: Scala, Erlang, Rust. Незважаючи на свою поширеність у більшості мов, у Python інструкції switch досі були відсутні. Однак, як ви незабаром побачите, тут нам пропонується набагато більше, ніж просто інструкції switch-case (що має на увазі match-case).

Трохи про pattern matching

Як написано в офіційній документації (PEP622), Python дуже часто вимагає перевірки даних на відповідність типів, звертатися до даних за індексом і до цих же даних застосовувати перевірку



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

на тип. Також часто доводиться перевіряти не тільки тип даних, а й кількість, що призводить до появи величезної кількості гілок if/else з викликом функцій isinstance, len і звернення до елементів за індексом, ключем або атрибутом. Саме для спрощення роботи та зменшення is/else було введено новий оператор match/case.

Дуже важливо не плутати pattern matching та switch/case, їхня головна відмінність полягає в тому, що pattern matching – це не просто оператор для порівняння деякої змінної зі значеннями, це цілий механізм для перевірки даних, їх розпакування та управління потоком виконання.

Приклад 1. Найпростіший приклад - це порівняння певної змінної зі значеннями (спочатку розглянемо, як це було б із if/else):

```
day = input('Введіть день тижня: ')
if day == 'понеділок':
   print('Початок робочого тижня')
elif day == 'вівторок':
   print('Другий робочий день')
elif day == 'середа':
   print('Середина тижня')
elif day == 'четвер':
   print('Передостанній робочий день')
elif day == 'п’ятниця':
   print('Останній робочий день тижня')
elif day == 'субота' or day == 'неділя':
   print('Сьогодні вихідний')
else:
   print('Немає такого дня тижня')
print()
```

Приклад 2. Застосуємо конструкцію match/case:

```
day = input('Введіть день тижня: ‘)
# Конструкція match/case:
match day:
   case 'понеділок':
      print('Початок робочого тижня')
   case 'вівторок':
      print('Другий робочий день')
   case 'середа':
      print('Середина тижня')
   case 'четвер':
      print('Передостанній робочий день')
   case 'п’ятниця':
      print('Останній робочий день тижня')
   case 'субота' | 'неділя':
      print('Сьогодні вихідний')
   case _:
      print('Немає такого дня тижня')
```

Тим не менш, за допомогою інструкцій match-case ми позбавляємося повторення day ==, що підвищує чистоту коду при тестуванні на відповідність багатьом різним умовам.

PEP 635 має чудові приклади інструкцій match-case, що підвищують читаність коду.



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

## Закріплення матеріалу

- Що таке алгоритм?
- Що таке алгоритм, що розгалужується?
- Як створювати розгалуження в програмах на Python?
- Як виглядає оператор if у Python? Охарактеризуйте його основні секції.
- Навіщо потрібний оператор pass?
- Навіщо потрібний оператор розгалуження з кількома умовами?
- Що таке умовний вираз?
- Як значення не-булевих типів можуть бути розглянуті як логічні значення і навіщо?

## Додаткове завдання

Завдання 1
Напишіть програму-калькулятор, в якій користувач зможе обрати операцію, ввести необхідні числа та отримати результат. Операції, які необхідно реалізувати: додавання, віднімання, множення, ділення, зведення в ступінь, квадратний корінь, кубічний корінь, синус, косинус та тангенс числа.

Завдання 2
Напишіть програму для перевірки введеного числа на ознаку парності. Якщо число парне/непарне – вивести відповідне повідомлення на екран. Для опису цього алгоритму використовувати тернарний оператор.

Завдання 3
Написати програму визначення днів тижня. Дані про день тижня вводяться користувачем з клавіатури. Якщо будній день - виводити на екран повідомлення "Сьогодні на роботу", у вихідні дні - "Сьогодні вихідний", в інших випадках - "Такого дня не існує".

## Самостійна діяльність учня

Завдання 1
Напишіть програму, яка запитує у користувача його ім'я, якщо воно збігається з вашим, видає певне повідомлення.

Завдання 2
Напишіть програму, яка обчислює значення функції $y$ = cos(3x), де - $\pi$ <= x <= $\pi$.

Завдання 3
Напишіть програму, яка розв'язує квадратне рівняння $ax^2 + bx + c = 0$ у дійсних числах. На відміну від аналогічної вправи з минулого уроку, програма повинна видавати повідомлення про відсутність дійсних коренів, якщо значення дискримінанта $D = b^2 - 4ac$ негативне, єдине рішення $x = -\frac{b}{2a}$, якщо він дорівнює нулю, або два корені $x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$, якщо він позитивний.

## Рекомендовані ресурси

Документація з 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

Статті у Вікіпедії про ключові поняття, розглянуті на цьому уроці
https://uk.wikipedia.org/wiki/Алгоритм
https://uk.wikipedia.org/wiki/Умовний_перехід



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