

Тема: Створення алгоритмів і програм з розгалуженням

Мета:

**Формування предметних компетентностей:**

уміння давати визначення понять “поліваріантне розгалуження”, “оператор вибору”;

розрізняти повну та скорочену форми оператора вибору;

складати, налагоджувати, тестувати програми з поліваріантним розгалуженням.

**Розвиток ключових компетентностей:**

*математична компетентність* (уміння застосовувати математичні методи для вирішення прикладних завдань у різних сферах діяльності);

*основні компетентності у природничих науках і технологіях* (уміння аналізувати інформаційні процеси, що відбуваються в живій природі, суспільстві та техніці; уміння будувати інформаційні моделі реальних об’єктів і процесів);

*інформаційно-цифрова компетентність* (уміння використовувати методи обробки інформації та чисельні методи для розв’язування прикладних задач);

*уміння вчитися впродовж життя* (уміння співвідносити свої дії з плановими результатами, здійснювати контроль своєї діяльності в процесі досягнення результату, визначати способи дій у межах запропонованих умов).



Теоретичний матеріал



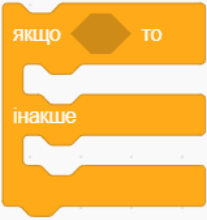
Сьогодні на уроці познайомимося ще з одним видом алгоритмів, які називаються алгоритмами з розгалуженням.

Ситуація, коли заздалегідь відома послідовність необхідних дій, зустрічаються вкрай рідко. У житті часто доводиться приймати рішення в залежності від ситуації. Якщо йде дощ, ми беремо з собою парасольку й надягаємо плащ; якщо спекотно, надягаємо легкий одяг. Зустрічаються й більш складні умови вибору. У деяких випадках від обраного рішення залежить подальша доля людини.

Команда розгалуження може бути повною і неповною.

Неповна форма команди розгалуження використовується тоді, коли необхідно виконувати дію тільки в разі дотримання умови. Якщо умова не дотримується, то команда розгалуження завершує свою роботу без виконання дії.

Працює повне розгалуження відповідно до схеми. Спочатку перевіряється умова. Якщо вона істинна, то виконується та послідовність інструкцій, яка знаходиться між ключовими словами `if` і `else`, або якщо і інакше (будемо називати цю послідовність інструкцій `if`-блоком). Якщо ж умова помилкова, то замість цього блоку інструкцій виконується та послідовність, яка йде після ключового слова `else`, інакше (`else`-блок). Після завершення виконання `if`-блоку або `else`-блоку, тобто незалежно від умови, починають виконуватися інструкції, наступні після `else`-блоку.

	
	<pre>if умова перша:     дія перша else:     дія друга</pre>









А як Пітон визначає, де закінчується `else`-блок і починаються ті інструкції, які будуть виконані незалежно від перевірки умов, тобто в якому місці наші шляхи виконання зливаються? Це визначається за величиною відступу: перед усіма інструкціями в `if`-блоці і `else`-блоці слід поставити один або кілька пробілів, виділяючи за допомогою зсуву вправо блок від решти програми, при тому, що число прогалін перед усіма інструкціями в блоці має бути однаковим. Перша ж інструкція після `else`-блоку повинна починатися з тієї ж позиції, що і ключові слова `if` і `else`. І ні в якому разі не слід забувати про двокрапки після інструкцій `if` і `else`.

**Умовний оператор (оператор розгалуження)** - це оператор, у якому здійснюється вибір однієї з двох дій у залежності від виконання або невиконання умови.

Як записувати умови. У найпростішому випадку умови мають такий вигляд:



## вираз1 умова вираз2


де вираз1 і вираз2- деякі арифметичні вирази (тобто змінні, арифметичні оператори, виклики функцій, тобто все, що зустрічалося нам в попередніх аркушах), а умова може бути наступним оператором порівняння:

	Опис	
	менше	<
	більше	>
	менше або дорівнює	<=
	більше або рівно	>=
	дорівнює	==
	не дорівнює	!=

Наприклад, умова  $x = 2 ** (0.5)$  означає “значення змінної  $x$  не менш кореня з 2”, а умова  $2 * x != y$  означає “подвоєне значення змінної  $x$  не дорівнює значенню змінної  $y$ ”.

У неповному розгалуженні може бути відсутнім ключове слово `else`, інакше з подальшим `else`-блоком.

	
---	--

	<code>if умова:     дія</code>
---	------------------------------------

На цьому уроці ми спробуємо навчитися створити програму, яка буде обчислювати математичні операції та програму, яка буде малювати рівносторонню геометричну фігуру при певній умові.

### **Задача №3**

Дано два цілих числа. Знайти суму, різницю, добуток і частку цих чисел.

Розглядаючи дану задачу на першому етапі, нам потрібно визначити, що дано і що потрібно визначити. А також визначити тип даних, з яким будемо працювати.

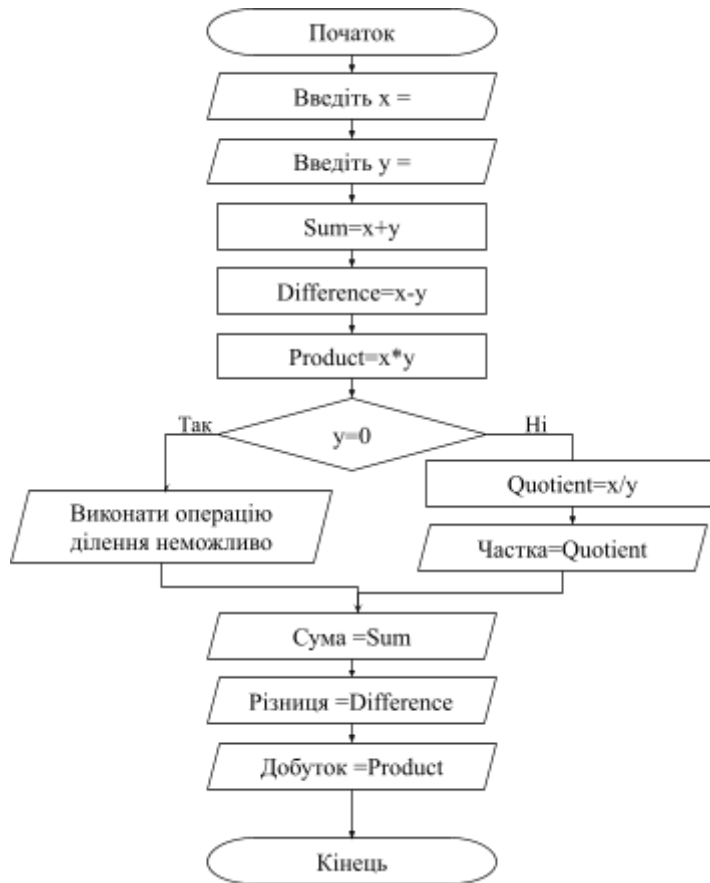
У даній задачі у нас є два числа, значення яких користувач вводить самостійно. На попередньому уроці ці два числа не могли бути нулем. Тепер користувач може ввести значення нуль. З'являється ситуація, коли перше число буде ділитися на нуль, тому нам потрібно врахувати такий випадок.

За змінні візьмемо  $x$ ,  $y$ , які будуть мати тип даних цілі числа.




На другому етапі будемо блок-схему. Для побудови блок-схеми будемо використовувати повне розгалуження.

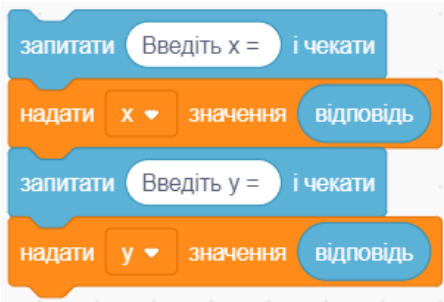
Останній третій етап створення програми.

Блок-схема



Далі спробуємо написати програму на Python. Спочатку подивимось, як команди Scratch замінюються на команди Python

	
<p>Першою командою, яка буде відповідати блок-схемі “Початок”</p>	
<p>У Scratch з групи “Події” -  . Ця команда нам потрібна буде для запуску програми,</p>	<p>Дана команда в Python зараз нам не потрібна. Програму написати можна відразу.</p>
<p>Далі за блок-схемою бачимо, що нам потрібно, щоб користувач надав змінним x, y значень. Використовуємо команди, такі ж самі, як і на попередньому уроці.</p>	



```
x=int(input("Введіть x = "))
y=int(input("Введіть y = "))
```

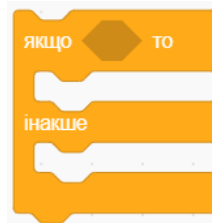
Далі повертаємося до блок-схеми. Потрібно зробити обчислення суми, різниці, добутку цих чисел. Частку даних будемо виконувати пізніше. Використовуємо команди, такі ж самі, як і на попередньому уроці.



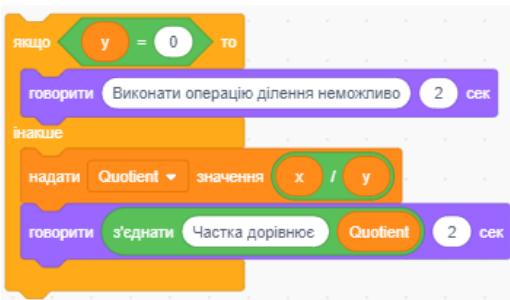
```
Sum=x+y
Difference=x-y
Product=x*y
```

Тепер нам потрібно перевірити, чи можливо поділити на число. Виконати операцію ділення неможливо при діленні на нуль.

У Scratch для перевірки умови потрібно використати з групи "Керування" команду для повного розгалуження



Використавши дану команду, отримаємо



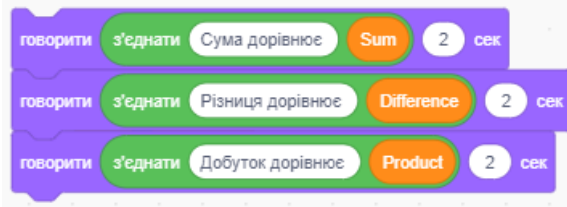
У Python перевірки умови потрібно використати умову

```
if умова перша:
    дія перша
else:
    дія друга
```

Використовуючи умову отримуємо

```
if y==0:
    print("Виконати операцію ділення неможливо")
else:
    Quotient=x/y
    print("Частка =",Quotient)
```

Далі повертаємося до блок-схеми. Тепер потрібно вивести інші результати обчислення на екран. Використовуємо команди, такі ж самі, як і на попередньому уроці.



```
print("Сума =", Sum)
print("Різниця =", Difference)
print("Добуток =", Product)
```

Повертаючись до блок-схеми, потрібно завершити програму.

Для завершення виконання програми використовують команду з групи "Керування"



У Python у лінійному алгоритмі окрему функцію для завершення програми прописувати не потрібно.

З'єднавши усі блоки команд, вийде повноцінна програма, в якій користувач вводить два числа, а програма виводить результати обчислень.

	<pre> x=int(input("Введіть x = ")) y=int(input("Введіть y = ")) Sum=x+y Difference=x-y Product=x*y if y==0:     print("Виконати операцію ділення неможливо") else:     Quotient=x/y     print("Частка =",Quotient) print("Сума =",Sum) print("Різниця =",Difference) print("Добуток =",Product) </pre>
--	--

Тепер переходимо до іншого прикладу.

#### Задача №4

Програма запитує у користувача намалювати квадрат чи ні. Якщо користувач вводить слово “так”, то програма малює квадрат червоного кольору. Якщо користувач вводить слово “ні” - нічого не виконує.

Розглядають дану задачу на першому етапі, нам потрібно визначити, що дано й що потрібно зробити.

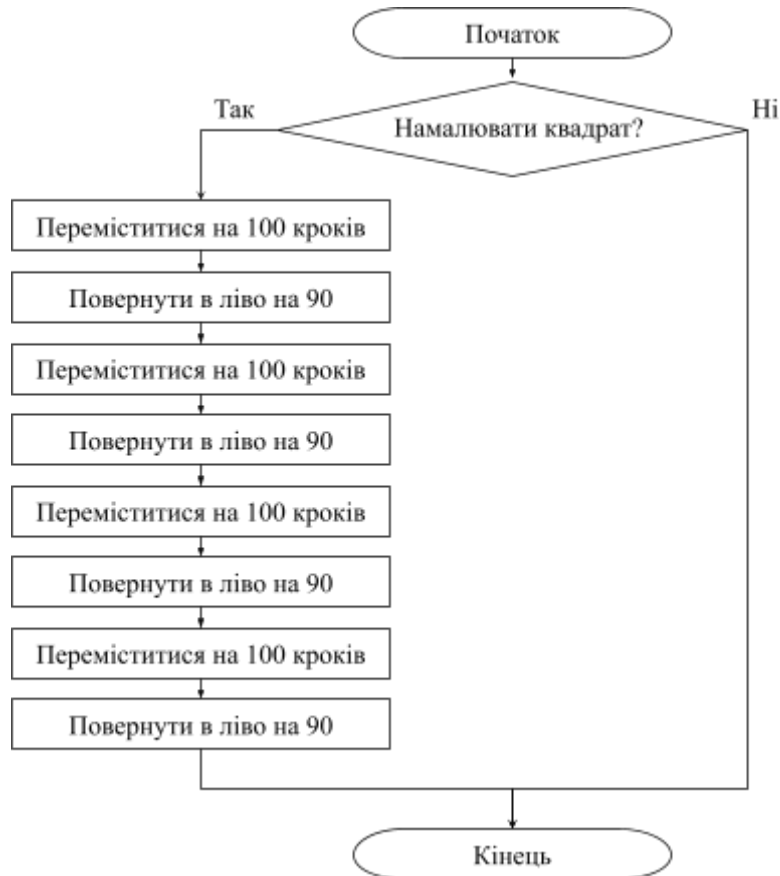
У даній задачі у нас є кут повороту та довжина сторони, значення яких прописано в програмі. Змінні в даній задачі використовувати не будемо. Потрібну довжину та кут повороту будуть вноситися відразу в команди.

На другому етапі будемо блок-схему. Дана схема допоможе визначити, які команди нам потрібні та в якому порядку потрібно буде їх виконати.

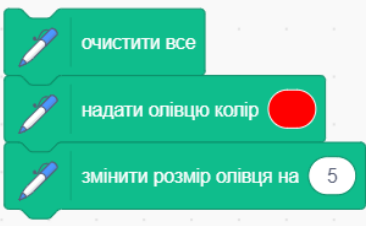
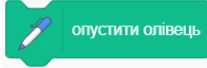
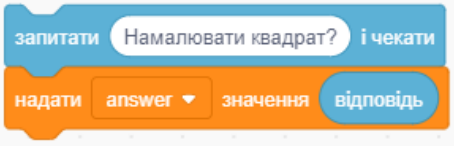

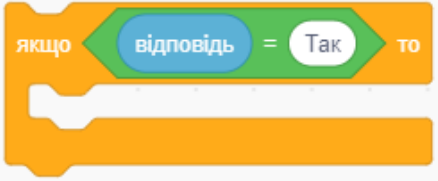


Останній третій етап - створення програми. Для цього в Scratch нам знадобляться команди груп: Олівець, Події, Керування, Датчики, Оператори, Рух.

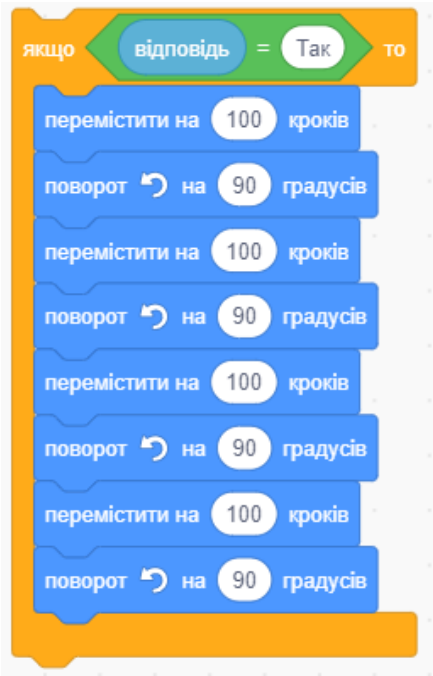
Блок-схема



	
<p>Спочатку потрібно додати бібліотеку малювання та прописати першу команду.</p>	
<p>У Scratch потрібно додати розширення “Олівець” Та додати першу команду</p> 	<p>Додаємо першу команду в програму <code>import turtle</code></p>
<p>Далі необхідно обрати колір олівця, товщину, очистити поле для малювання.</p>	

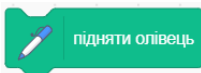
	<pre>turtle.clear() turtle.pencolor("red") turtle.width(5)</pre>
<p>Для того щоб почати малювати, потрібно опустити олівець.</p>	
<p>Команда  дозволить опустити олівець, який буде потім малювати.</p>	<p>Команда <code>turtle.down()</code> дозволить опустити олівець, який буде потім малювати.</p>
<p>Тепер створюємо умову при якій буде будуватися квадрат. Нам потрібно перевірити, що відповідь користувач на запитання “Намалювати квадрат?”, якщо “Так”, то будуюмо, якщо “Ні”, то нічого не виконуємо. Але спочатку потрібно буде створити змінну “answer”, яка буде зберігати відповідь користувача.</p>	
<p>Команди  допоможуть запитати та присвоїти змінній відповідь. Далі використовуємо команду  для створення умови. Використовуючи дану команду отримаємо </p>	<p>Команди <code>answer=str(input("Намалювати квадрат?"))</code> допоможуть запитати та присвоїти змінній відповідь. Для цієї змінної потрібно використати рядковий тип даних. Далі використовуємо команду <code>if умова:</code> дія. Використовуючи дану команду отримаємо <code>if answer=="Так":</code></p>

Для побудови квадрата нам потрібно побудувати чотири сторони, як на попередньому уроці. Дані команди помістити таким чином, щоб при відповіді “Так” вони виконувалися.



```
if answer=="Так":  
    turtle.forward(100)  
    turtle.left(90)  
    turtle.forward(100)  
    turtle.left(90)  
    turtle.forward(100)  
    turtle.left(90)  
    turtle.forward(100)  
    turtle.left(90)
```

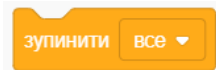
Після побудови всіх сторін потрібно підняти олівець.



```
turtle.up()
```

У кінці потрібно завершити програму.

Для завершення виконання програми використовують команду з групи “Керування”



У Python у лінійному алгоритмі окрему функцію для завершення програми прописувати не потрібно.

З'єднавши усі блоки команд, які розбирали при створенні, вийде програма побудови червоного квадрату.

The Scratch script consists of the following blocks:

- коли** **натиснуто** (when green flag clicked)
- очистити все** (clear all)
- надати олівцю колір** (set pen color to red)
- задати розмір олівця** (set pen size to 5)
- опустити олівець** (put pen down)
- запитати** "Намалювати квадрат?" **і чекати** (ask "Draw a square?" and wait)
- якщо** **відповідь** **=** **Так** **то** (if answer is Yes then)
  - перемістити на** 100 **кроків** (move 100 steps)
  - поворот** **на** 90 **градусів** (turn 90 degrees)
  - перемістити на** 100 **кроків** (move 100 steps)
  - поворот** **на** 90 **градусів** (turn 90 degrees)
  - перемістити на** 100 **кроків** (move 100 steps)
  - поворот** **на** 90 **градусів** (turn 90 degrees)
  - перемістити на** 100 **кроків** (move 100 steps)
  - поворот** **на** 90 **градусів** (turn 90 degrees)
- підняти олівець** (pick up pen)
- зупинити** **все** (stop all)

```
import turtle
turtle.clear()
turtle.pencolor("red")
turtle.width(5)
turtle.down()
answer=str(input("Намалювати квадрат?"))
if answer=="Так":
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
turtle.up()
```