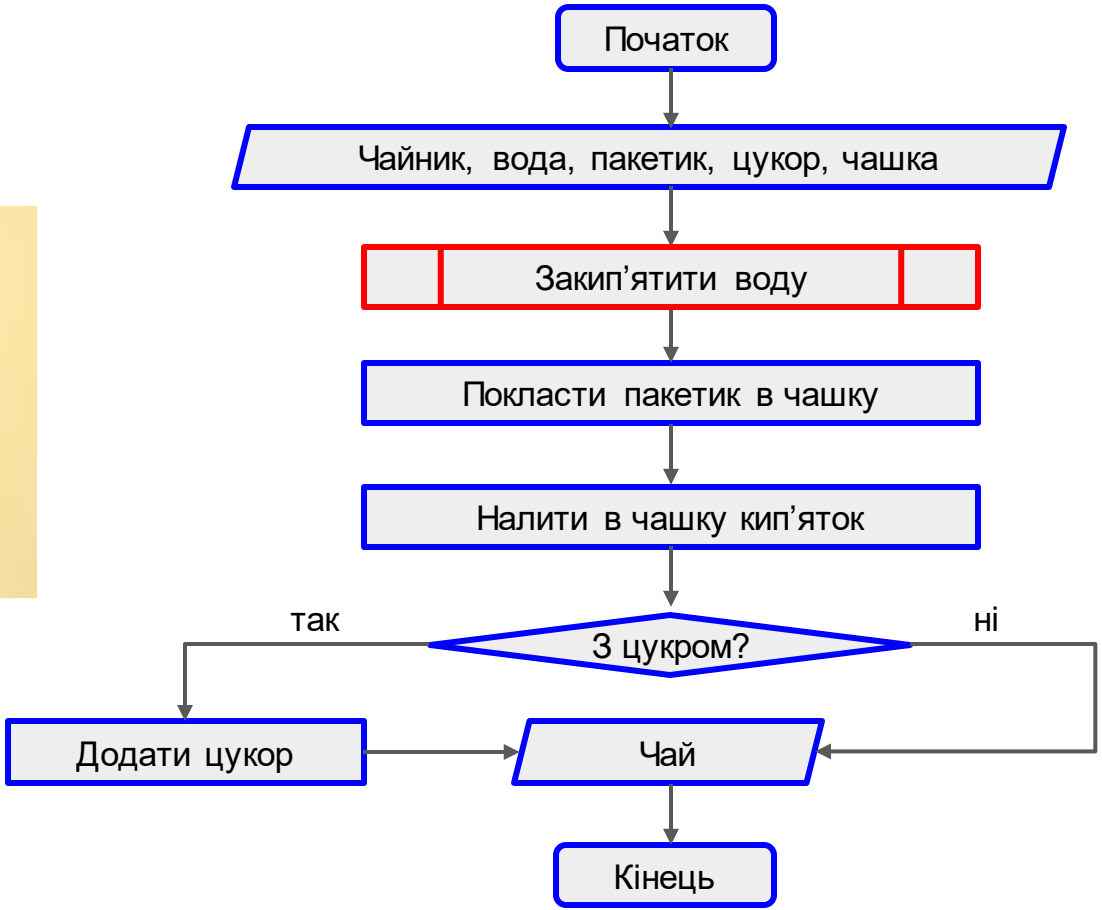
ПРОГРАМУВАННЯ (PYTHON)

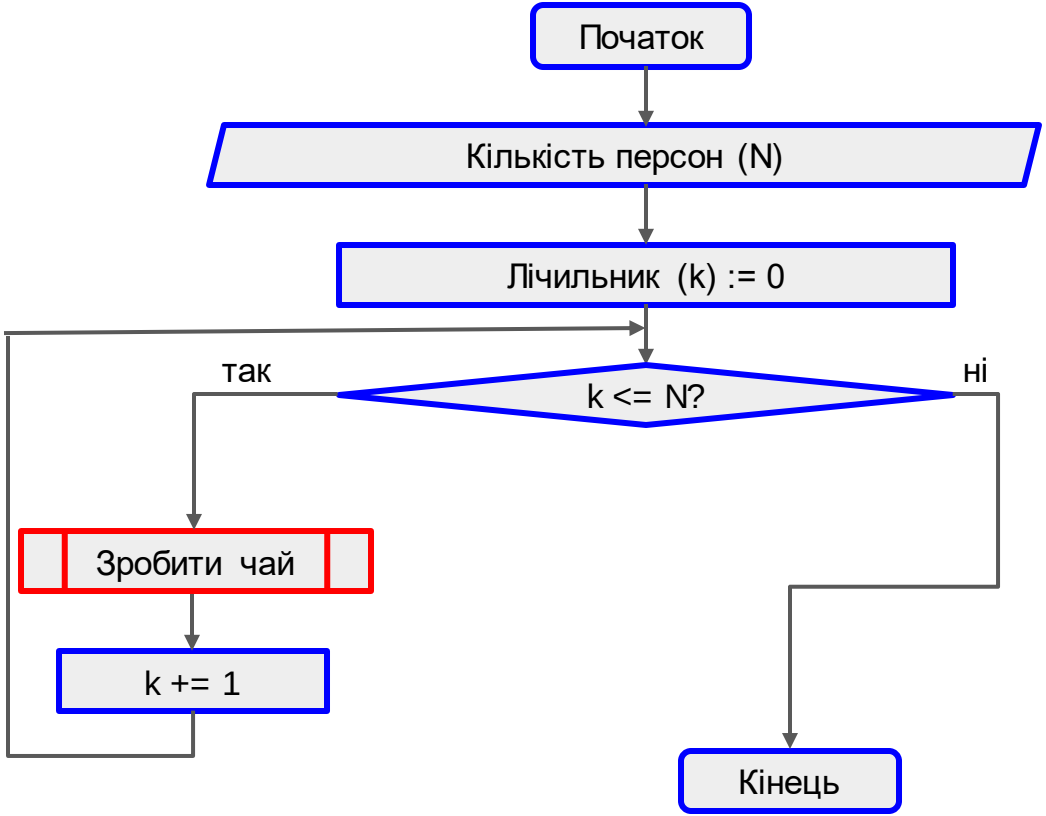
РОЗГАЛУЖЕННЯ ТА ЦИКЛИ

ПОСЛІДОВНІСТЬ (ЛІНІЙНІ АЛГОРИТМИ)

РОЗГАЛУЖЕННЯ



ЦИКЛ


Початок
Кількість персон (N)
Лічильник (k) := 0
так
k <= N?
ні
Зробити чай
k += 1
Кінець

Логічний тип

Логічний тип **bool** складається з двох взаємовиключних значень, які можуть бути записані кількома способами, але найвживанішими є пара **True** і **False**. Вони трактуються як "правда" і "неправда", "так" і "ні" тощо.

В Python логічний тип є підмножиною типу **int**, оскільки його значення поводяться так само, як цілі числа **1** і **0**, лишень відображаються як **True** і **False** для кращого сприйняття:

```
print(True + False - 3)          # -2
```

Логічні вирази

Умови, які перевіряються під час виконання програми, записуються у формі **логічних виразів**, результатом яких є одне із значень логічного типу.

Залежно від цього значення інтерпретатор виконує ту чи іншу послідовність інструкцій.

Оператори порівняння

Прості логічні вирази зазвичай утворюються з використанням операторів порівняння:

== (дорівнює); != (не дорівнює); < (менше); > (більше);
<= (менше або дорівнює); >= (більше або дорівнює)

```
3 == 5                  # False
3 != 3.0                # False
3 < 5                   # True
5 >= 5                  # True
```

Порівняння дійсних чисел, через їх наближене подання у пам'яті комп'ютера, може видати неочікувані результати:

```
print(0.1**3 == 0.001)              # False
```

Подумайте, який вихід?

Порівняння значень різних типів у загальному випадку не допускається — спроба порівняти число і рядок викличе помилку:

```
5 > '2'
TypeError: '>' not supported between instances of 'int' and 'str'
```

Але:

```
5 != '5'      # True (число 5 і символ '5' – не одне й те саме)
```

Деякі особливості перевірки на істинність:

- **будь-яке число (крім 0) та будь-який непорожній об'єкт інтерпретується як True;**
- **число 0, будь-який порожній об'єкт і спеціальний об'єкт None інтерпретується як False;**
- **Оператори and і or повертають операнди!!!**

З операторів порівняння можна утворювати ланцюжки:

```
1 != 2 <= 2+1 > 1.5 # True

'A' < 'B' > 'C' # False (так, рядки теж можна порівнювати)
```

Списки та кортежі (як і рядки) порівнюються поелементно, але рекурсивно і можуть повертати помилку, словники не порівнюються (лише == і !=), оператори типу < для множин означають **підмножина**.

Для об'єднання результатів перевірок використовуються логічні оператори and (і), or (або) і not (заперечення):

```
not (2 > 3)                                    # True
2 > 3 or 3 < 4                                 # True
2 + 2 == 4 and 2 * 2 != 4          # False
```

!!! Додаткові можливості оператора or

```
X = A or B or C or None          # X - перший непорожній об'єкт
```

Це можна використати для присвоєння значення за замовчуванням:

```
X = A or default
```

```
if f1() or f2(): …
```

Якщо f1() поверне True, то f2() вже не буде запускатися.

Логічні вирази можуть використовувати оператор перевірки на входження in, який використовується для перевірки належності об’єкта певному складеному типу:

```
'u' in 'Ukraine'                    # False
'rain' in 'Ukraine'                 # True
'U' not in 'Ukraine'                # False
```

Інструкція if

```
if логічний_вираз:
    …
    …
```

```
# програма виводить повідомлення, якщо введена літера є
# у слові “інформатика”

word = 'інформатика'
if input('Введіть літеру: ') in word:
    print('Введена Вами літера є у слові, word)
```

Конструкції if...else і if...elif...else

```
if логічний_вираз:
    …
    …
else:
    …
    …
```

```
if логічний_вираз_1:
    …
    …
elif логічний_вираз_2:
    …
    …
elif логічний_вираз_3:
    …
else:
    …
```

```
# програма виводить інформацію про парність/непарність введеного числа


number = int(input('Введіть число: '))
if number % 2 == 0:
    print('Число', number, 'парне')
else:
    print('Число', number, 'непарне')
```

Подумайте, чи можна записати логічний вираз `number % 2 == 0` якось інакше?

```
# програма порівнює два введені числа


x = int(input('Введіть перше число: '))
y = int(input('Введіть друге число: '))

if x > y:
    print('Перше число більше за друге')
elif x < y:
    print('Перше число менше за друге')
else:
    print('Числа однакові')
```

Відступи (пробіли або символи табуляції на початку рядка) є частиною синтаксису Python.

Усі інструкції в межах одного блоку повинні мати однаковий відступ, який має бути оформлений або пробілами, або символами табуляції (змішування не допускається)

Оскільки складені інструкції можуть містити в собі інші складені інструкції, то неправильне оформлення відступів може призвести до некоректних результатів (семантичні помилки). Наприклад, наведений нижче синтаксично правильний код видасть некоректний результат для віку 10 років.

```
age = int(input('Введіть Ваш вік: '))
if age < 18:
    print('Ви неповнолітні')
    if age < 5:
        print('Ви не школяр')
    else:
        print('Ви повнолітні')
```

Який? чому? як правильно оформити відступи?

В окремих випадках допускається записувати умовну інструкцію в один рядок:

```
if hours > 24: print('Пройшло більше доби')
```

або

```
x = 1 if y>3 or y!=5 else 0
```

Домашнє завдання: напишіть багаторядкові еквіваленти цих однорядкових інструкцій

Цикли

Більшість прикладних задач передбачають багаторазове виконання одних і тих самих інструкцій.

У програмуванні багатократно виконувані послідовності інструкцій називаються **циклами**, а кожне повторення — **ітерацією**.

Розрізняють два види циклів — з наперед відомою і невідомою (можуть виконуватися безкінечно або припинятися за виконання певної умови) кількістю ітерацій. Кожному з цьох видів відповідає своя керівна конструкція Python.

Функція range()

range() – універсальний інструмент генерування цілих чисел.

Зазвичай використовується для організації циклів з відомою кількістю ітерацій.

Може приймати один, два або три аргументи.

У випадку з одним аргументом функція генерує цілі числа від 0 до значення аргумента, не включаючи останнього:

```
range(5)                # генерує числа 0, 1, 2, 3, 4
```

Якщо функції передати два аргументи, перший вважатиметься початком відліку:

```
range(-3, 3)            # генерує числа -3, -2, -1, 0, 1, 2
```

Третім аргументом можна задавати крок:

```
range(-10, 10, 3)    # генерує числа -10, -7, -4, -1, 2, 5, 8
```

Крок може бути від'ємним:

```
range(5, 0, -1)              # генерує числа 5, 4, 3, 2, 1
```

Цикл for … in …

Цикл for … in зазвичай використовуються при відомій кількості повторень і має такий синтаксис:

```
for <ім'яЗмінної> in range(<аргументи>):
    <блок інструкцій>
```

Як і у випадку з if, це складена інструкція — заголовок циклу закінчується двокрапкою, а інструкції тіла циклу оформляються з відступами.

Змінна, яка використовується у заголовку циклу, почергово приймає значення, згенеровані функцією range()

```
# сума та добуток перших n натуральних чисел

n = int(input('Введіть n: ')

suma, dobutok = 0, 1 # початкові значення суми і добутку
for i in range(1,n+1): # надаємо і значень від 1 до n
    suma = suma + i # додаємо до суми значення і
    dobutok = dobutok * i # множимо добуток на значення і

print(suma, dobutok)
```

У циклах часто використовуються так звані комбіновані присвоєння (у випадку додавання/віднімання їх ще називають операторами інкременту/декременту):

```
x += a          # аналог x = x + a
x -= a          # аналог x = x - a
x *= a          # аналог x = x * a
x /= a          # аналог x = x / a
x %= a          # аналог x = x % a
...
```

Домашнє завдання: Виправте код з попереднього слайду, використавши комбіновані присвоєння

Конструкція **for...in** також часто застосовується для:

- генерування списків, словників і множин;
- порядкового обходу файлів;
- посимвольного обходу рядків;
- поелементного обходу колекцій.

Цикл while

На відміну від циклу for, тіло якого виконується заздалегідь відому кількість разів, інструкція while дає змогу організувати цикл, кількість ітерацій якого непередбачувана.

Найпростіший синтаксис цієї складеної інструкції такий:

```
while <логічний вираз>:
    <блок іструкцій>
```

Блок інструкцій виконується, поки значення логічного виразу дорівнює True. Принаймі одна з інструкцій повинна змінювати значення деякої змінної, яка входить у логічний_вираз і може надати йому значення False. Інакше цикл буде нескінченним.

```
# генерування випадкових цифр, допоки не буде згенеровано 0

from random import randint
number = randint(0,9) # допоміжна інструкція (перша цифра)
while number:                          # аналог while number != 0:
    print(number) # друкуємо поточну цифру
    number = randint(0,9) # генеруємо наступну цифру
```

Допоміжна інструкція потрібна для того, щоб увійти у цикл, інакше під час виконання наступного рядка виникне помилка, оскільки інтерпретатор не зможе обчислити значення логічного виразу, бо нічого не знатиме про змінну number.

Інструкція break

Інструкція break використовується для примусового виходу з циклу. Тобто, її виконання змушує інтерпретатор перейти до першої інструкції після циклу.

```
from random import randint
while True: # можна while 1:
    number = randint(0,9) # генеруємо цифру
    if not number: # те саме, що if number == 0:
        break
    print(number)
```

Домашнє завдання: Що буде надруковано, якщо останній рядок коду оформити без відступу?

Інструкція continue

Інструкція continue змушує інтерпретатор пропустити усі інструкції тіла циклу, що йдуть після неї і перейти до заголовку циклу (наступної ітерації).

```
# генерування випадкових парних цифр, допоки не 0
from random import randint
number = 1 # будь-яка цифра для того, щоб увійти в цикл

while number: # поки цифра не ноль
    number = randint(0,9) # генеруємо цифру
    if number % 2: # якщо цифра непарна,
        continue # то пропускаємо друк
    print(number)
```

Блок else

Цикли можуть містити необов'язковий блок **else**, який виконується тільки у випадку непримусового завершення циклу, тобто його є зміст використовувати лише з інструкцією break, інакше він виконуватиметься завжди.

```
# чи введене ціле число просте
x = int(input('Введіть ціле число: '))
for i in range(2, x//2 + 1):
    if x % i == 0:
        print('Число не є простим')
        break
else:
    print('Число просте')
```

```
# чи натуральне число є сумою квадратів двох натур. чисел
n = int(input('Введіть натуральне число: '))
limit = round(n**0.5) + 1 # максимальне складове число
label = 0 # мітка, яка вказує на негативну відповідь
for n1 in range(1, limit): # зовнішній цикл
    for n2 in range(1, limit): # вкладений цикл
        if n1**2 + n2**2 == n:
            label = 1 # змінюємо мітку на позитивну
            Break # і виходимо з внутрішнього циклу
    if label: # якщо мітка позитивна,
        break # то виходимо і з зовнішнього циклу
if label:
    print(f'{n} = {n1}^2 + {n2}^2')
else:
    print(f'Число {n} не є сумою квадратів')
```