Тема 4
Умови та розгалуження

Умови

Програмісти повинні вміти навчити комп'ютер або програму приймати рішення. Це дуже важливо для створення складних, але більш якісних програм. Ці рішення приймає комп'ютер, але саме програміст повинен написати код для цього. Можна написати програму, яка буде реагувати на вхідні дані від користувача під час її виконання та видавати різні результати для різних вхідних даних, наприклад дій користувача в грі. Цей процес стає можливим за допомогою умов.

Уявіть, за ігровим сценарієм для кожної події існує час або рівень, якого гравець повинен досягти, щоб перейти на наступний рівень.

```
#Кваліфікація пройдена = 100
if TriathlonPoints<100:
    print('Спробуйте ще раз')
if TriathlonPoints>=100:
    print('Готуйтеся до наступної події')
```

*Ви використали слово **if** і двокрапку (:).*
*За словом **if** слідує умова.*
*У цьому випадку **TriathlonPoints<100**.*
*Комп'ютер порівнює значення в змінній **TriathlonPoints**. Якщо воно менше 100, він виконує команди, що стоять нижче двокрапки (:) – умовне виконання.*

У цьому прикладі дві умови: **TriathlonPoints<100** і **TriathlonPoints>=100**. Однак немає необхідності перевіряти виконання другої умови (**TriathlonPoints>=100**). Можна змінити код так, щоб він мав такий вигляд:

```
#Кваліфікація пройдена = 100
if TriathlonPoints<100:
    print('Спробуйте ще раз')
else:
    print('Готуйтеся до наступної події')
```

else (інакше) робить ваш код набагато коротшим.

Коли ви хочете присвоїти дані змінній, використовуйте (=), наприклад **RunningScore=60**. Але (=) можна використовувати й для порівняння, коли порівнюєте змінні чи змінну зі значенням, як-от **RunningScore==60**.

Вкладені умови

Іноді потрібно зробити вибір між більш ніж двома можливими випадками. У Python можна використовувати **elif**. Команда буде виконана, як тільки одна з умов стане істинною.

```
#Отримання медалі
if TriathlonPoints>=120:
    MedalName='Золото'
elif TriathlonPoints>=115:
    MedalName='Срібло'
elif TriathlonPoints>=110:
    MedalName='Бронза'
else:
    print('Наступного разу все вийде')
```


© Binary Logic та © Видавництво «Лінгвіст» . Усі права захищені. Жодна частина цієї публікації не може бути відтворена, збережена в пошуковій системі або передана в будь-якій іншій формі будь-якими способами без письмового дозволу видавця. За жодних обставин ніяка частина цього матеріалу не може бути скопійована для перепродажу або для іншого використання.

Умовні та логічні оператори

Під час написання умов можна використовувати наведені нижче умовні та логічні оператори.

Умовні оператори

ОПЕРАТОР	ЗНАЧЕННЯ
==	Дорівнює
>	Більше ніж
<	Менше ніж
>=	Більше або дорівнює
<=	Менше або дорівнює
!=	Не дорівнює

Логічні оператори

ОПЕРАТОР	ЗНАЧЕННЯ
and	Обидві сторони повинні бути істинними
or	Одна зі сторін повинна бути істинною
not	Заперечує істинність

Цикли

Припустимо, у вас є список зі спробами гравців у стрибках у довжину, і треба відобразити найвищий стрибок без використання функції **max()**.

Ви створили змінну, надали їй початкове й кінцеве значення та дозволили комп'ютеру збільшити змінну. Кожного разу, коли змінна збільшується, комп'ютер виконує команди після оператора **for**.

```
LongJumpList=[6, 7.25, 5.5, 6.2]
BestTry=0
for i in range(0,4):
    if LongJumpList[i]>BestTry:
        BestTry=LongJumpList[i]
print('Ваш найкращий результат:', BestTry)
```

Цей цикл виконується чотири рази. Змінна **i** починається з **0** й закінчується значенням **3**.

Зазвичай приріст дорівнює **1**, але його можна змінити на будь-яке значення, визначивши крок у параметрах діапазону. Можна вказати навіть від'ємний приріст, щоб рахувати назад.

```
for i in range(25,1,-2):
    print(i)
```

Значення **–2** в параметрах діапазону задає приріст змінної **i**.

```
25
23
21
19
17
15
13
11
9
7
5
3
```

Цикл **for** використовують, коли знають, скільки разів будуть виконані команди. Кількість повторів указують за допомогою параметра **range**. Але іноді невідомо, скільки разів буде виконуватися цикл. У цьому випадку скористайтеся циклом **while**, який виконується до тих пір, поки виконується задана умова.

```
#Пароль повинен мати 6 символів
password=input('Введіть пароль: ')
while len(password)<6:
    print('Ваш пароль повинен мати 6 символів. Спробуйте ще раз.')
    password=input('Введіть пароль: ')
```

Всі команди після слова **while** будуть виконуватися до тих пір, поки кількість символів у змінній **password** буде меншою за 6.

© Binary Logic та © Видавництво «Лінгвіст» . Усі права захищені. Жодна частина цієї публікації не може бути відтворена, збережена в пошуковій системі або передана в будь-якій іншій формі будь-якими способами без письмового дозволу видавця. За жодних обставин ніяка частина цього матеріалу не може бути скопійована для перепродажу або для іншого використання.

За допомогою вкладених циклів можна значно скоротити код.

Виведемо на екран вкладені списки та їхні елементи.

```
list=[[1,2],['c','d'],[15,62,79]]
for i in list:
#виводить на екран елементи зовнішнього списку
    print(i)
    for j in i:
#виводить на екран елементи внутрішнього списку
        print(j)
```

```
[1, 2]
1
2
['c', 'd']
c
d
[15, 62, 79]
15
62
79
```

Оператор **break** дозволяє достроково перервати виконання поточного процесу, як-от цикли **for** або **while**, чи перервати виконання умови **if ... else**.

```
#Пароль повинен мати 6 символів
password=input('Введіть пароль: ')
while True:
    if len(password)<6:
        print('Ваш пароль повинен мати 6 символів. Спробуйте ще раз.')
        password=input('Введіть пароль: ')
    else:
        break
```

```
Введіть пароль: 321
Ваш пароль повинен мати 6 символів. Спробуйте ще раз.
Введіть пароль: 8hiO123ikj
```

Оператор **continue** повертає управління на початок циклу **for** або **while**. Він дозволяє «перестрибнути» поточний крок і перейти до наступної ітерації.

```
i = 0
while i < 9:
    i += 1
    if i == 5:
        print("Число 5 пропустимо")
        continue
        print("Це число ніхто не побачить")
    else:
        print("Поточний індекс: ", i)
```

```
Поточний індекс:  1
Поточний індекс:  2
Поточний індекс:  3
Поточний індекс:  4
Число 5 пропустимо
Поточний індекс:  6
Поточний індекс:  7
Поточний індекс:  8
Поточний індекс:  9
```

У програмуванні зазвичай існує багато способів виконання однієї і тієї самої дії. Іноді один спосіб кращий за інший. Продуктивність, розмір пам'яті та інші фактори можуть бути важливими. Вибір залишається за вами.

Практичне завдання

Завантажте файл-заготовку зі сторінки інтернет-підтримки та виконайте запропоновані завдання.

© Binary Logic та © Видавництво «Лінгвіст» . Усі права захищені. Жодна частина цієї публікації не може бути відтворена, збережена в пошуковій системі або передана в будь-якій іншій формі будь-якими способами без письмового дозволу видавця. За жодних обставин ніяка частина цього матеріалу не може бути скопійована для перепродажу або для іншого використання.