

Range, enumerate, try-except

Функції range(), enumerate()

Функція `range()` працює наступним чином: вона приймає від одного до трьох чисел та робить діапазон послідовних чисел. Якщо введене одне число, то границею автоматично стане 0. Якщо два числа – діапазон буде будуватися від першого до другого, при цьому перше число обов'язково повинне бути менше за друге. Третій параметр – це шаг (за замовченням він = 1).

Структура даних, яку створює функція, називається *діапазоном*. Ця структура не є змінною.

При написанні границь варто враховувати, що нижня границя входить до діапазону, а верхня – ні.

Так, `range(5, 8)` буде відповідати 5,6,7.

Функцію `range()` дуже ефективно поєднувати з функцією `len()`. За допомогою даної конструкції можна змінити кожен елемент списку за допомогою цикла `for`:

```
spis = [7, 3.42, -14, 0]
for i in range(len(spis)):
    spis[i] += 2
print(spis)
```

Результат виконання циклу: `[9, 5.42, -12, 2]`

Таким чином, цикл `for` перебирає не елементи списку, а, відповідно, їх індекси.

Якщо `range()` дає нам можливість оперувати індексами списку, то функція `enumerate()` генерує кортежі з парами індекс-елемент.

Цикл:

```
spis = [7, 3.42, -14, 0]
for i in enumerate(spis):
    print(i)
```

Результат виконання циклу:

```
(0, 7)
(1, 3.42)
(2, -14)
(3, 0)
```

Ці кортежі можливо розпаковувати. Для цього, зазвичай використовують дві змінні у заголовку циклу `for`:

Цикл:

```
spis = [7, 3.42, -14, 0]
for i, k in enumerate(spis):
    print(i, k)
```

Результат виконання циклу:

```
0 7
1 3.42
2 -14
3 0
```

Виключення в мові Python

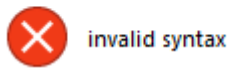
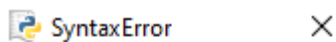
У будь-яких мовах бувають помилки. Помилки бувають *синтаксичні* (в результаті програма не працює взагалі) та *логічні* (в результаті програма робить не те, що було заплановано).

Виникнути помилка може з ряду причин.

Синтаксичні помилки (*виключення*) бувають наступні:

SyntaxError

Власне, це і є випадок, коли виявляється неправильне написання змінної (наприклад, вона починається з числа або з якогось спецсимволу).



ValueError

Виникає при спробі перевести змінну у неприпустимий для неї формат (наприклад, перевести строку з буквами у значенні у число).

```
Traceback (most recent call last):  
  File "C:/Users/Magnum/Desktop/ddd.py", line 2, in <module>  
    stre = int(stre)  
ValueError: invalid literal for int() with base 10: '2k'
```

ZeroDivisionError

Виникає при спробі поділити на нуль.

```
Traceback (most recent call last):  
  File "C:/Users/Magnum/Desktop/ddd.py", line 2, in <module>  
    stre = int(stre)/0  
ZeroDivisionError: division by zero
```

TypeError

Виникає при виконанні дії з операндами неприємлемого типу (наприклад, підвести число у степінь змінної, значення якої – строка).

```
Traceback (most recent call last):  
  File "C:/Users/Magnum/Desktop/ddd.py", line 2, in <module>  
    stre = stre/2  
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```


NameError

Виникає при спробі використання змінної, значення якої не вказувалось раніше.

```
Traceback (most recent call last):
  File "C:/Users/Magnum/Desktop/ddd.py", line 3, in <module>
    print(st)
NameError: name 'st' is not defined
```

IndentationError

Виникає при вказанні неправильного відступу.

 SyntaxError



unexpected indent

IndexError

Виникає при вказанні недійсного для списку або кортежу індексу.

```
Traceback (most recent call last):
  File "C:/Users/Magnum/Desktop/ddd.py", line 2, in <module>
    print(stre[5])
IndexError: list index out of range
```

KeyError

Виникає при вказанні неіснуючого ключа для словника.

```
Traceback (most recent call last):
  File "C:/Users/Magnum/Desktop/ddd.py", line 2, in <module>
    print(stre[5])
KeyError: 5
```

Якщо знати що означають виключення та вміти їх читати, то можна доволі швидко привести код у робочий стан.

Конструкція try-except

При виникненні виключення програма закривається аварійно. Так як Python виконується построчно, помилка може не бути виявлена одразу. На випадок, якщо є шанс виникнення якоїсь з помилок, існує конструкція *try-except*.

Вона складається з двох гілок:

Try – спробувати виконати ділянку коду. Якщо виникає помилка, то інтерпретатор автоматично переходить до гілки *except*. А якщо гілка *try* виконалась без помилок, то гілка *except* пропускається.

Приклад поєднання try-except з циклом while:

```
a = input('Введіть число ')
while type(a) == str:
    try:
        a = float(a)
    except:
        a = input('Введіть число, а не букву ')
```

Спочатку ми даємо можливість ввести число та отримуємо строку у будь-якому випадку. Після чого відбувається вход у цикл, так як в перший раз умова буде вірною у будь-якому випадку. В гілці `try` ми спробуємо перевести строку до числа, а якщо буде `ValueError` – примушуємо користувача вводити строку до тих пір, поки виключення не зникне. Таким чином ми виключили варіант аварійного закриття через користувача.

Можна зробити так, щоб гілка `except` реагувала на якийсь конкретний тип виключень. Тоді при різних помилках будуть активуватися різні прописані дії. Для позначення окремого типу виключення пишемо:

```
except ValueError:
```

Замість `ValueError` може стояти будь-який тип.

Контрольні питання:

1. Як створити діапазон?
2. Для чого може бути корисним знання синтаксичних помилок?
3. Як “перехватити” помилку?