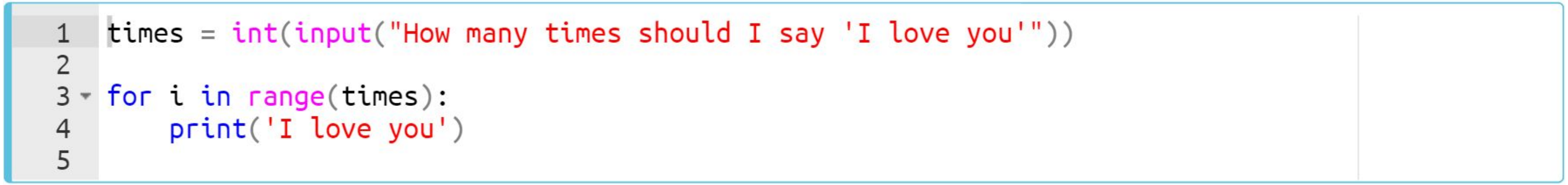
Operations with for Loop

Operations with the `for` Loop(review)

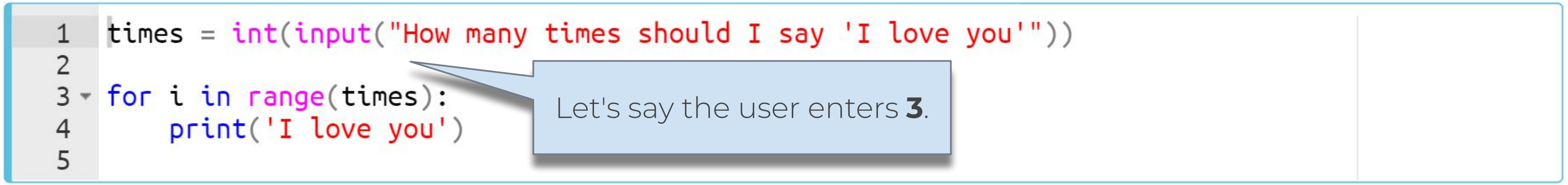
- In the following example, you'll get a **number** from the user and *print* **a sentence** the *number of times* we take from the user :



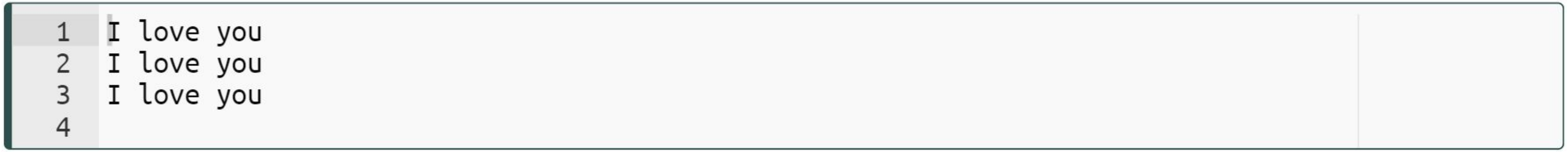
```python
times = int(input("How many times should I say 'I love you'"))

for i in range(times):
    print('I love you')

```

Operations with the for Loop(review)

- In the following example, you'll get a number from the user and print a sentence the number of times we receive from the user :



```python
times = int(input("How many times should I say 'I love you'"))

for i in range(times):
    print('I love you')

```



```
I love you
I love you
I love you

```

CLARUSWAY©
WAY TO REINVENT YOURSELF

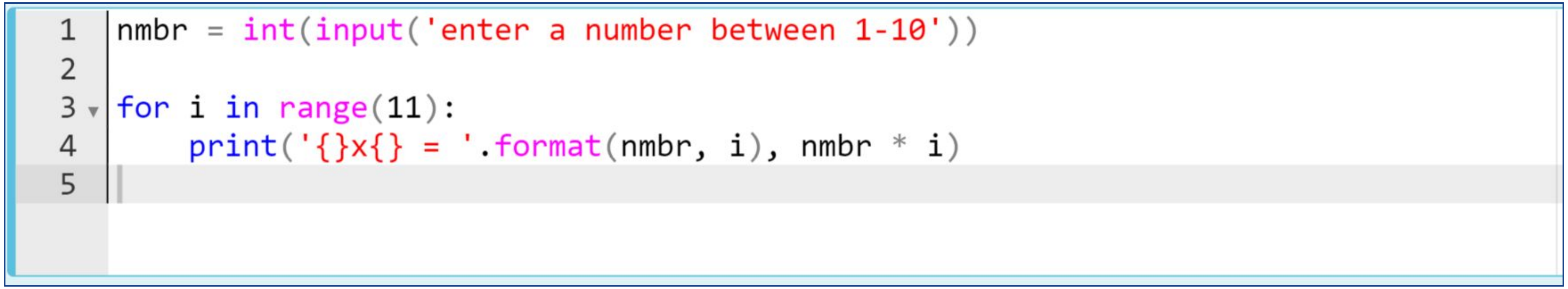
Operations with the `for` Loop

- **Task :** This time, write a code block that asks the user a number between 1 and 10 then puts that number into the multiplication table.
- For example, the output for 5 should be as follows :

```
5x0 =  0
5x1 =  5
5x2 =  10
5x3 =  15
5x4 =  20
5x5 =  25
5x6 =  30
5x7 =  35
5x8 =  40
5x9 =  45
5x10 =  50
```

Working with the Iterators

- **The output can be like :**



```
nmbr = int(input('enter a number between 1-10'))

for i in range(11):
    print('{}x{} = '.format(nmbr, i), nmbr * i)

```

Operations with the `for` Loop(review)

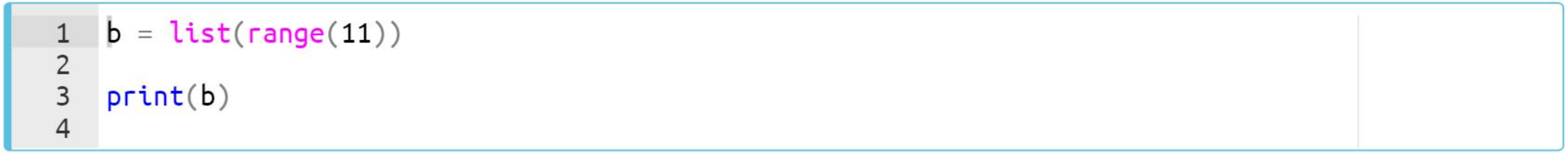
- Let's take a close look at the `range()` function.
  - As we stated before, the formula syntax of the `range()` function is :

```
range(start, stop, step)
```

$\underbrace{\texttt{start, stop, step}}_{\text{parameters}}$

Operations with the `for` Loop(review)

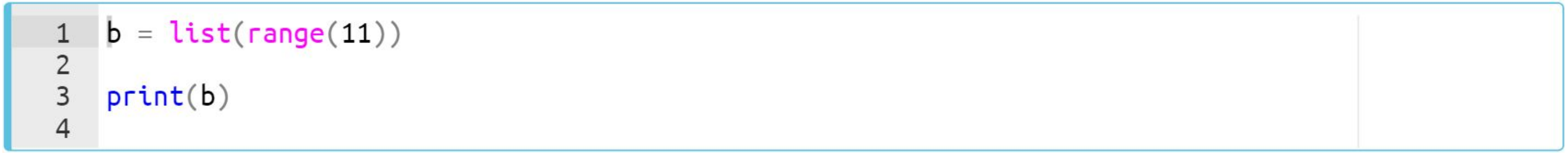
- (... continued)
  - Consider this example :



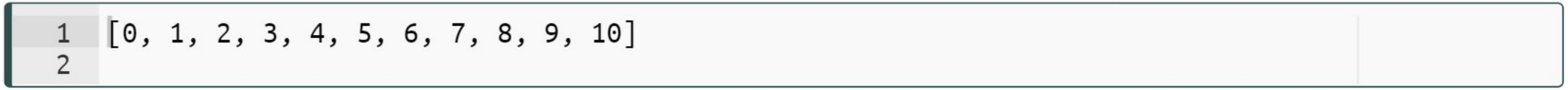
```python
b = list(range(11))

print(b)

```

Operations with the `for` Loop(review)

- Let's take a close look at the `range()` function.
  - It creates an iterable sequence of numbers. And it can be simply converted into the `list`, `set`, and `tuple`.
  - Consider this example :



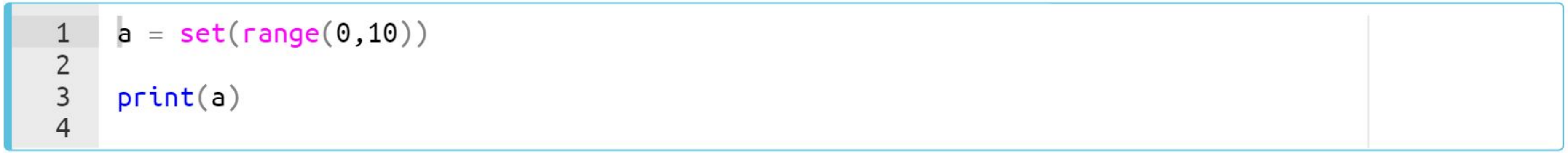
```python
b = list(range(11))

print(b)

```



```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

```

Operations with the `for` Loop(review)

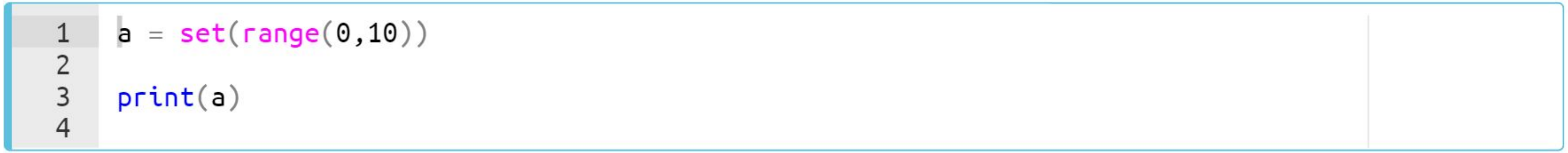
- (… continued)
  - Here's the other examples :



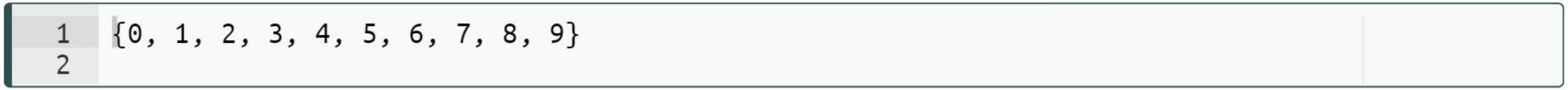
```python
a = set(range(0,10))

print(a)

```

Operations with the `for` Loop(review)

- (... continued)
  - Here's the other examples :



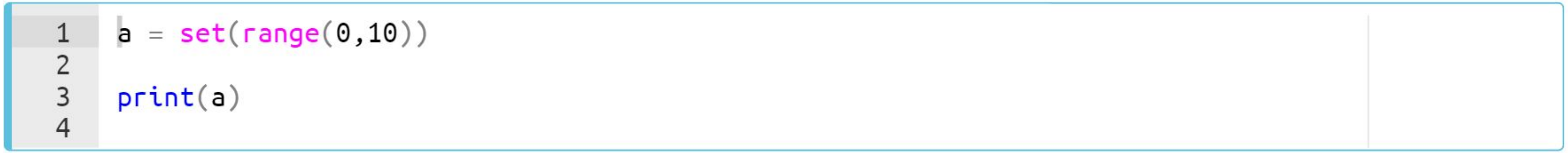
```python
a = set(range(0,10))

print(a)

```



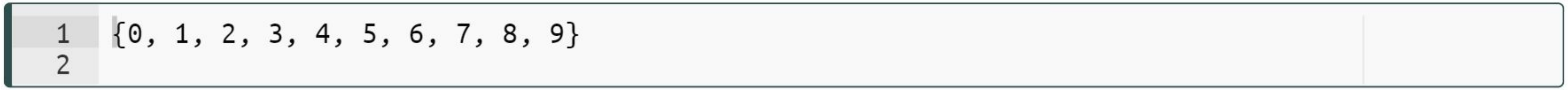
```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

```

Operations with the `for` Loop(review)

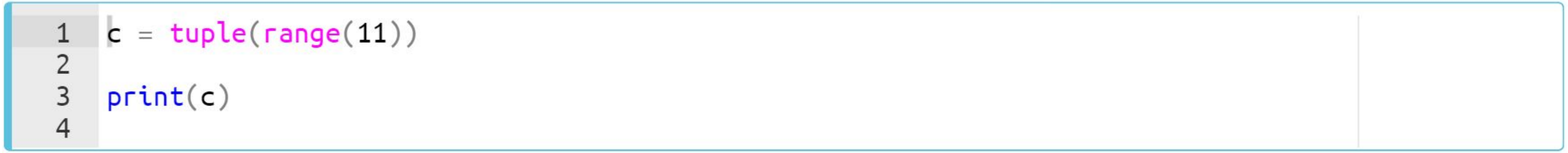
- (... continued)
  - Here's the other examples :



```python
a = set(range(0,10))

print(a)
```



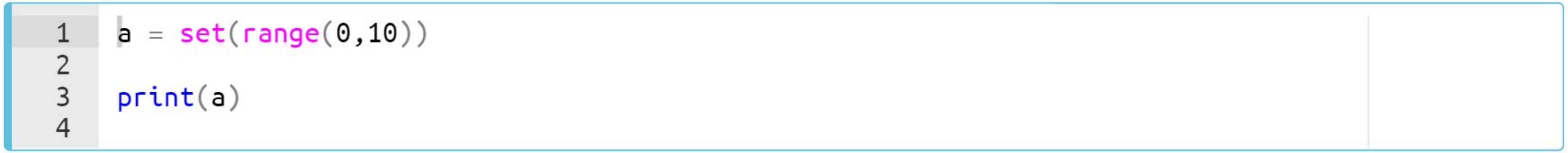
```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```



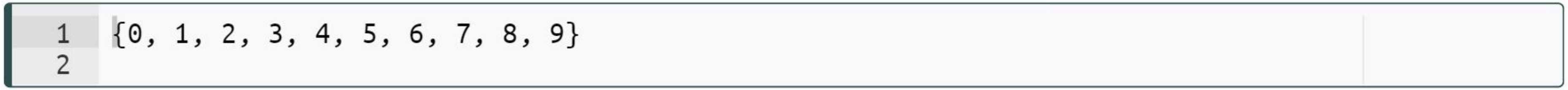
```python
c = tuple(range(11))

print(c)
```

Operations with the for Loop(review)

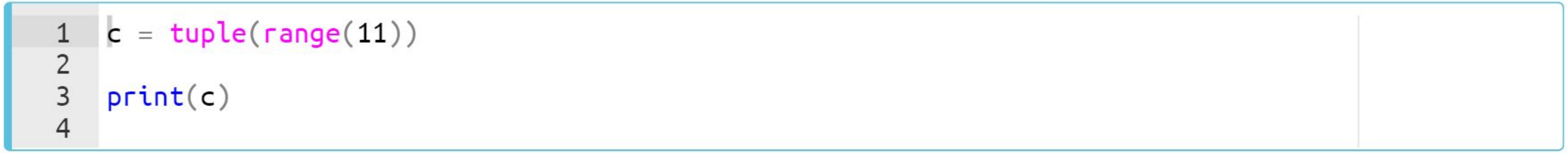
- (... continued)
  - Here's the other examples :



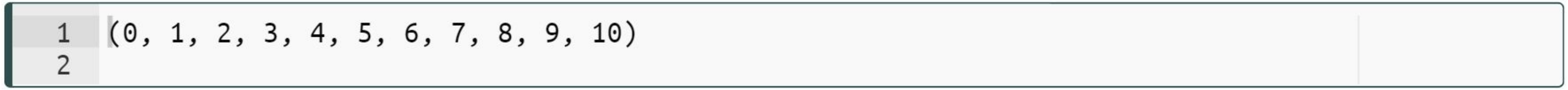
```
a = set(range(0,10))

print(a)

```



```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

```



```
c = tuple(range(11))

print(c)

```



```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

```

Operations with the for Loop(review)

- An asterisk 👉 * separates the elements of the iterables.
  - Let's take a look at an example of the range() function with starred 👉 * expression :

```
print(range(5))  # it will not print the numbers in sequence

print(*range(5))  # '*' separates its elements

```

What is the output? Try to figure out in your mind...

Operations with the `for` Loop(review)

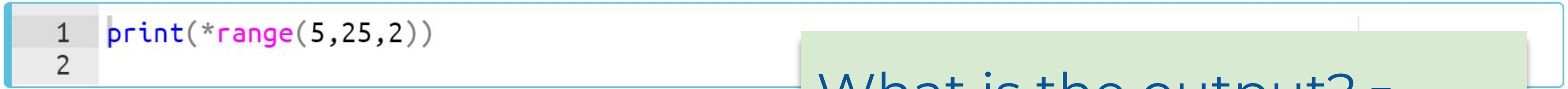
- (... continued)
  - Let's take a look at an example of the `range()` function with starred 👉 `*` expression :

```
print(range(5))   # it will not print the numbers in sequence

print(*range(5))   # '*' separates its elements

```

```
range(0, 5)
0 1 2 3 4

```

Operations with the `for` Loop(review)

- (... continued)
  - Here's another example of the `range()` function with starred 👉 `*` expression :

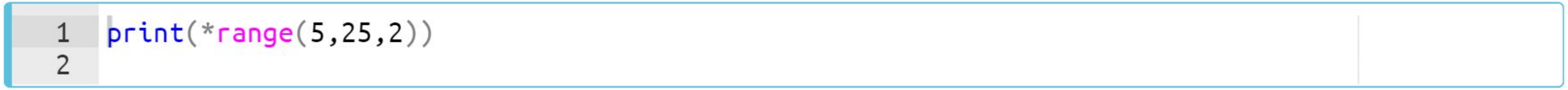


```
print(*range(5,25,2))

```

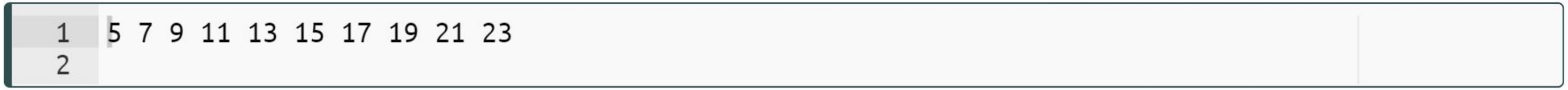
What is the output? Try to figure out in your mind...

Operations with the for Loop(review)

- (... continued)
  - Here's another example of the range() function with starred 👉 * expression :



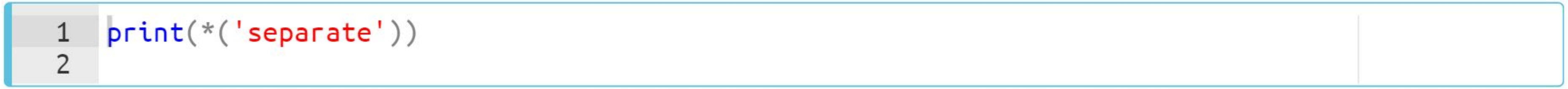
```
print(*range(5,25,2))
```



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

Operations with the `for` Loop(review)

- (... continued)
  - Starred 👉 `*` expression can also be used to separate the other **iterable** objects. Such as `str` :



```python
print(*('separate'))
```

Operations with the `for` Loop(review)

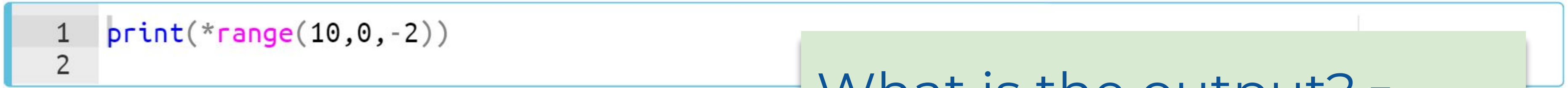
- (… continued)
  - Starred 👉 `*` expression can also be used to separate the other **iterable** objects. Such as `str` :

```python
print(*('separate'))
```

```
s e p a r a t e
```

Operations with the for Loop(review)

- (... continued)
  - You can create reverse sequence numbers using a negative `step`.

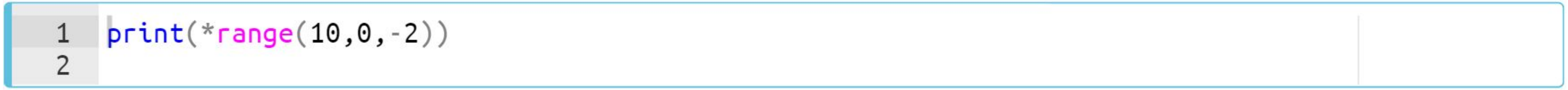


```
print(*range(10,0,-2))

```

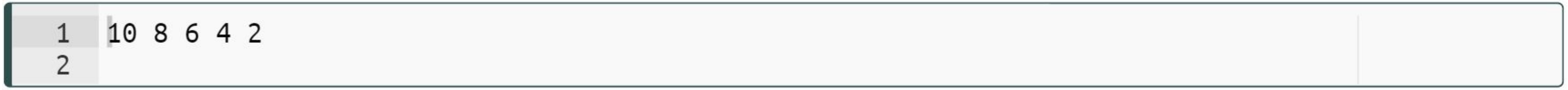
What is the output? Try to figure out in your mind...

Operations with the `for` Loop(review)

- (... continued)
  - You can create reverse sequence numbers using a negative `step`.



```python
print(*range(10,0,-2))
```



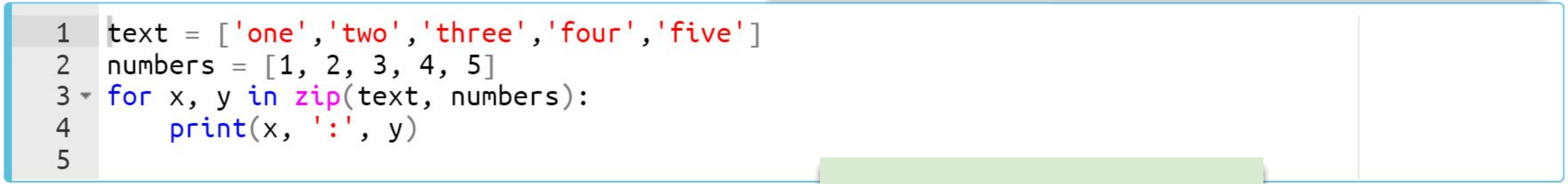
```
10 8 6 4 2
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

Operations with the `for` Loop(review)

- Multiple variables in `for` loop.
  - Examine this example carefully :

```
zip(iterator1, iterator2, ...)
```



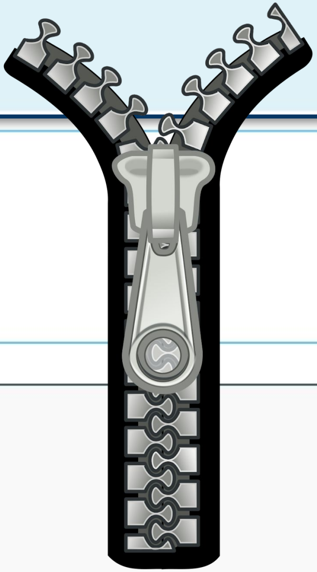
```
text = ['one','two','three','four','five']
numbers = [1, 2, 3, 4, 5]
for x, y in zip(text, numbers):
    print(x, ':', y)
```

Use your IDEs

Operations with the for Loop(review)

Tips :

- zip() function make an iterator that aggregates elements from each of the iterables.



```
text = ['one','two','three','four','five']
numbers = [1, 2, 3, 4, 5]
for x, y in zip(text, numbers):
    print(x, ':', y)

```

```
one : 1
two : 2
three : 3
four : 4
five : 5

```

CLARUSWAY©
WAY TO REINVENT YOURSELF

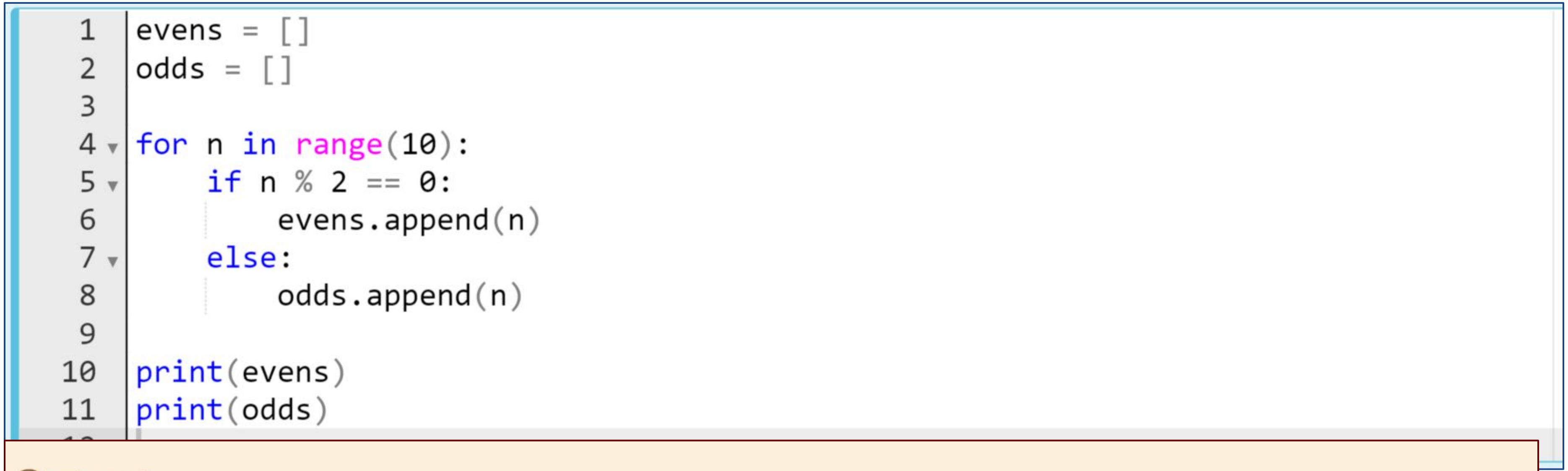
Operations with the `for` Loop

- **Task** : **Python Program to collect the odd and even numbers in two different `lists`.**
  - Write a program to choose and collect the even and odd numbers (1 to 10) in two different `list`.
  - Print the result such as :

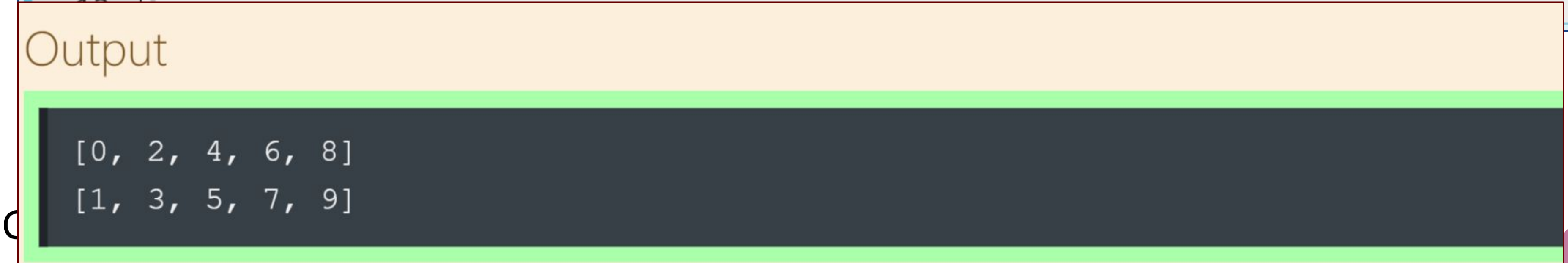
```
evens: [0, 2, 4, 6, 8]
odds : [1, 3, 5, 7, 9]
```

Operations with the `for` Loop

- **The code might be like** :



```python
evens = []
odds = []

for n in range(10):
    if n % 2 == 0:
        evens.append(n)
    else:
        odds.append(n)

print(evens)
print(odds)
```



Output

```
[0, 2, 4, 6, 8]
[1, 3, 5, 7, 9]
```

Operations with the `for` Loop

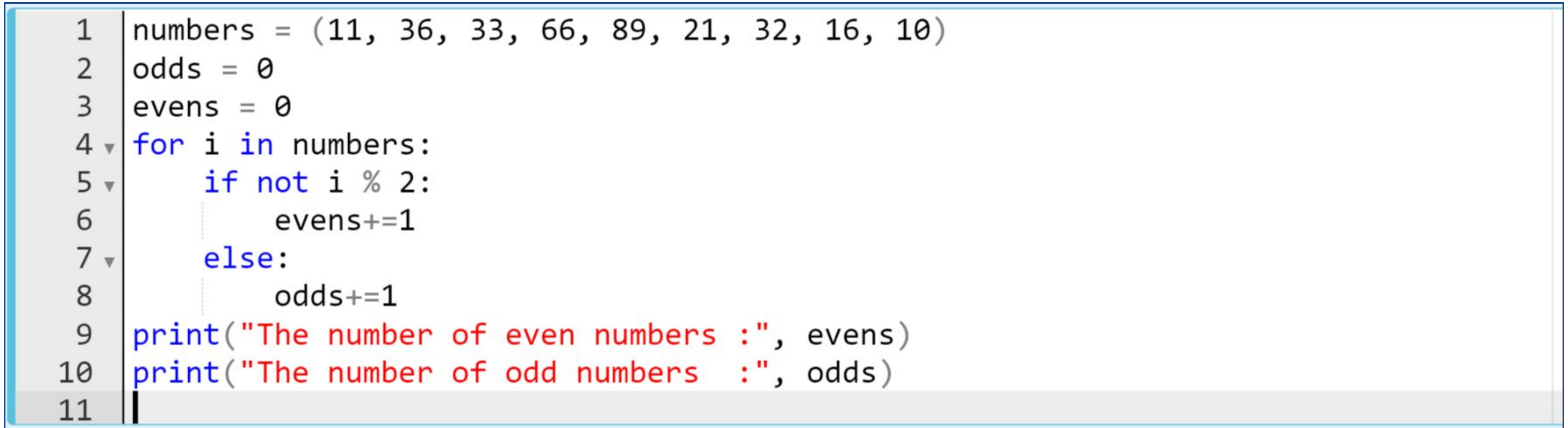
- **Task** : **Python Program to sum the amount of odd and even numbers in a `tuple`/`list`.**
  - Write a code that counts the odd and even numbers in a given `list` or `tuple`.
  - Print the result such as :

```
example list: [11, 2, 24, 61, 48, 33, 3]
example output : The number of even numbers : 3
                 The number of odd numbers : 4
```

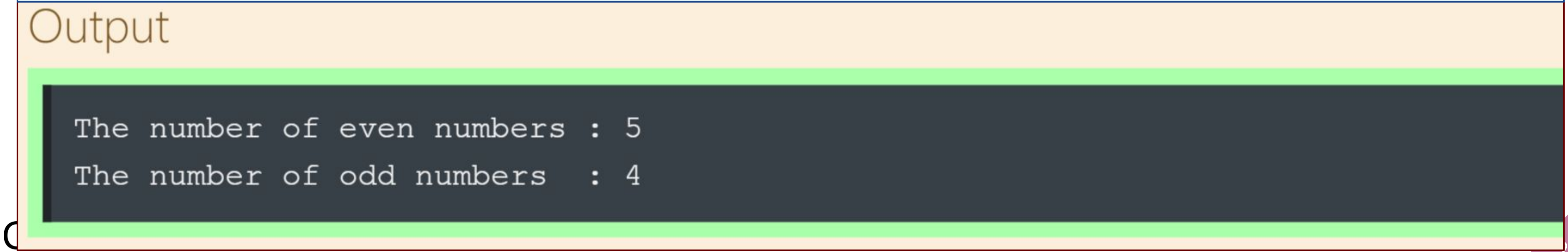
CLARUSWAY©
WAY TO REINVENT YOURSELF

Operations with the for Loop

- **The code might be like :**



```python
numbers = (11, 36, 33, 66, 89, 21, 32, 16, 10)
odds = 0
evens = 0
for i in numbers:
    if not i % 2:
        evens+=1
    else:
        odds+=1
print("The number of even numbers :", evens)
print("The number of odd numbers  :", odds)
```



Output

```
The number of even numbers : 5
The number of odd numbers  : 4
```

Operations with the `for` Loop

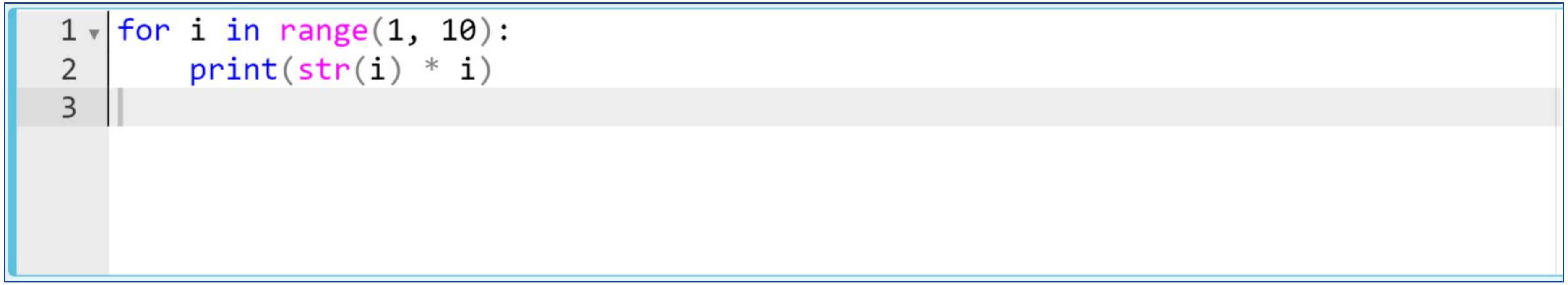
- **Task** : **Python Program to print out the numbers.**
  - Using the `for` loop, print the numbers from **1** to **9** as many as it is and get the following output.

```
1
22
333
4444
55555
666666
7777777
88888888
999999999
```

CLARUSWAY
WAY TO REINVENT YOURSELF

Operations with the for Loop

- **The code might be like :**



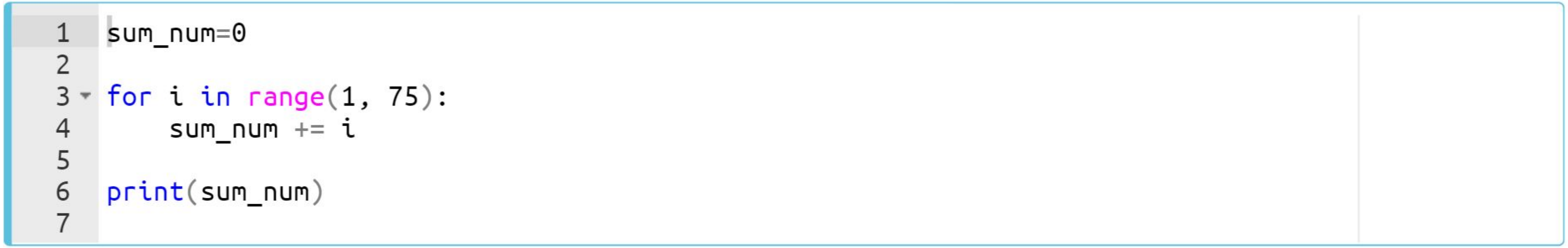
```python
for i in range(1, 10):
    print(str(i) * i)

```

Operations with the `for` Loop

- **Task** : **Python Program to sum of the numbers from 1 to 74**
  - Get the output of **2775** as a sum of the numbers between **1** - **74** (including).
  - Use `for` loop to make this calculation.

Operations with the for Loop

- **The code might be like :**



```python
sum_num=0

for i in range(1, 75):
    sum_num += i

print(sum_num)
```

Nested for Loop

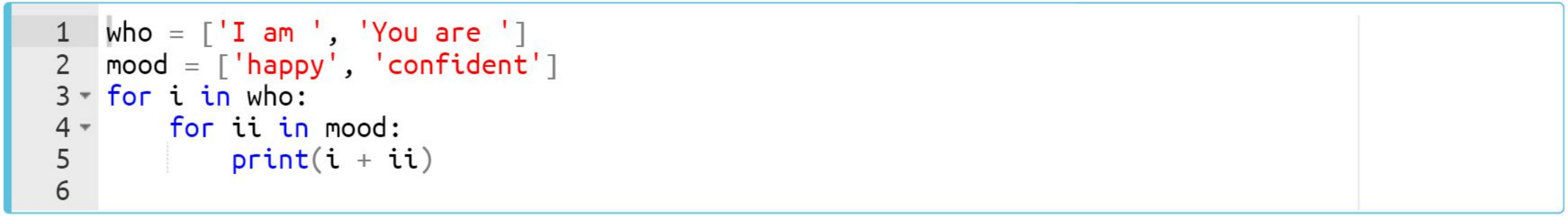
Nested for Loop (review)

- Simple structure of the nested for loops look like :

```
for variable1 in iterable1:
    for variable2 in iterable2:
        body
```

Nested for Loop (review)

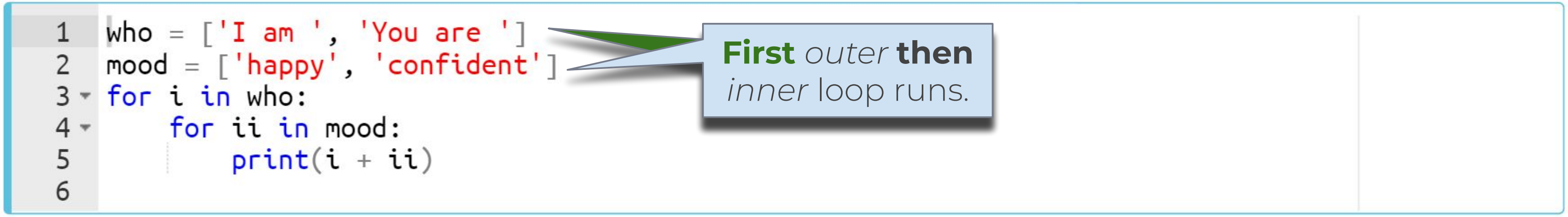
- Consider this example of the nested for loop :



```
who = ['I am ', 'You are ']
mood = ['happy', 'confident']
for i in who:
    for ii in mood:
        print(i + ii)

```

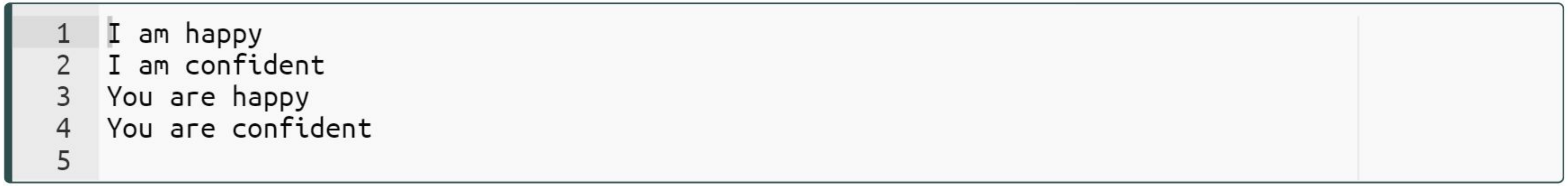
Nested for Loop

- Consider this example of the nested for loop :



```
who = ['I am ', 'You are ']
mood = ['happy', 'confident']
for i in who:
    for ii in mood:
        print(i + ii)
```

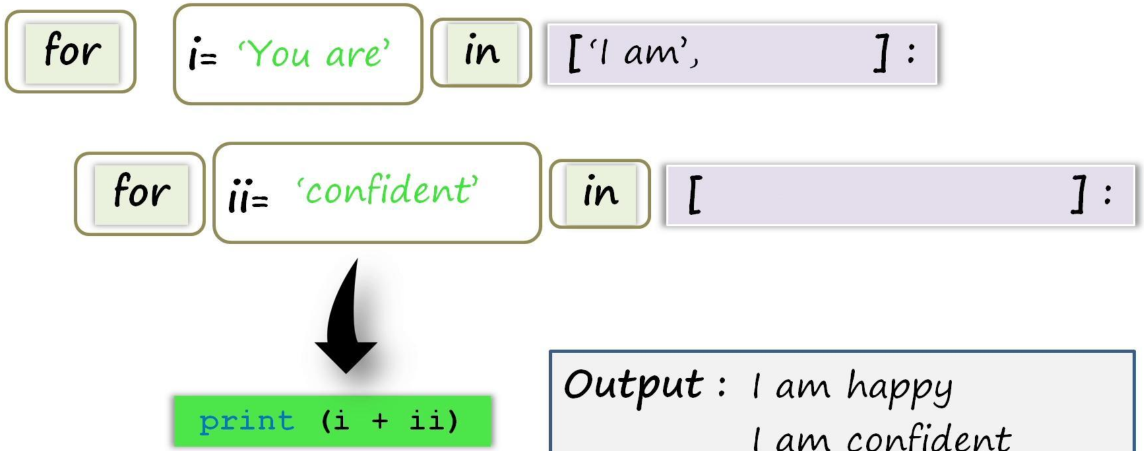
First *outer* **then** *inner* loop runs.



```
I am happy
I am confident
You are happy
You are confident
```

Nested for Loop (review)

- You can follow the animated diagram of this nested for loop for a better understanding.



```
print (i + ii)
```

Output : I am happy
I am confident
You are happy

Nested `for` Loop

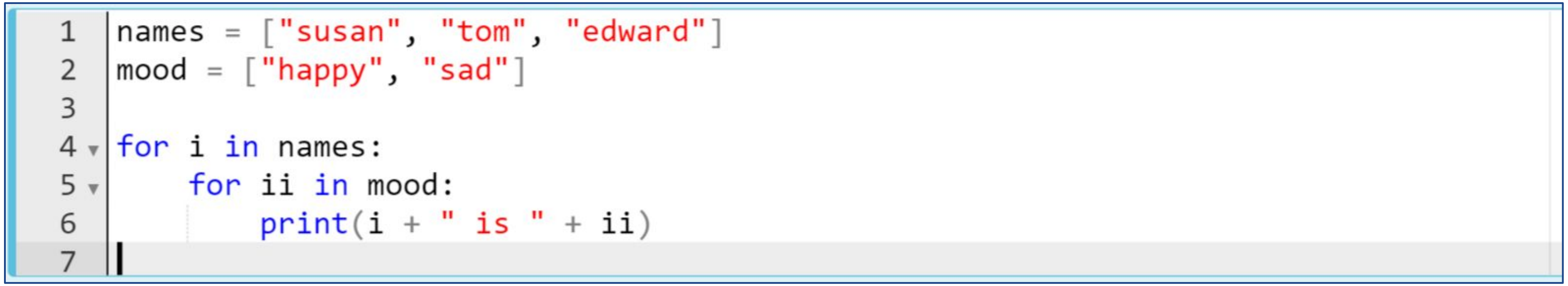
- **Task** : **Concatenation string elements from two separate `lists`.**
  - Write a code that takes string elements one by one and prints a sentence using nested `for` loops :
  - The given lists and sample outputs are :

```
names = ["susan", "tom", "edward"]
mood = ["happy", "sad"]
example output : susan is happy
                 susan is sad
                 tom is happy
                      .
                      .
```

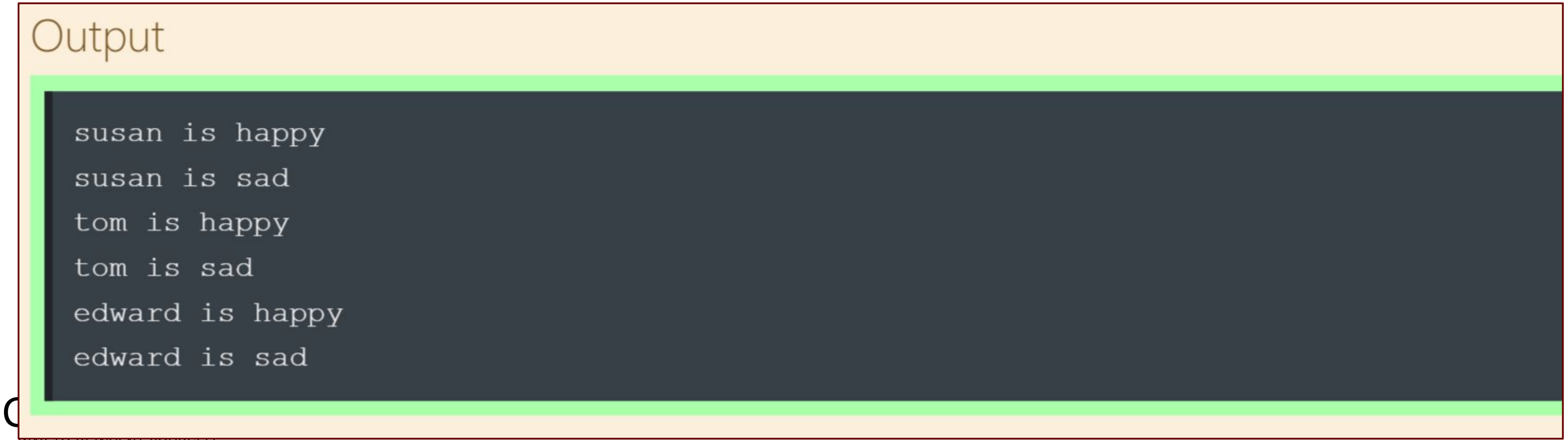
CLARUSWAY©
WAY TO REINVENT YOURSELF

Nested for Loop

- **The code might be like :**



```python
names = ["susan", "tom", "edward"]
mood = ["happy", "sad"]

for i in names:
    for ii in mood:
        print(i + " is " + ii)

```



Output

```
susan is happy
susan is sad
tom is happy
tom is sad
edward is happy
edward is sad
```

List Comprehensions (List Generators)

Written in a long form;

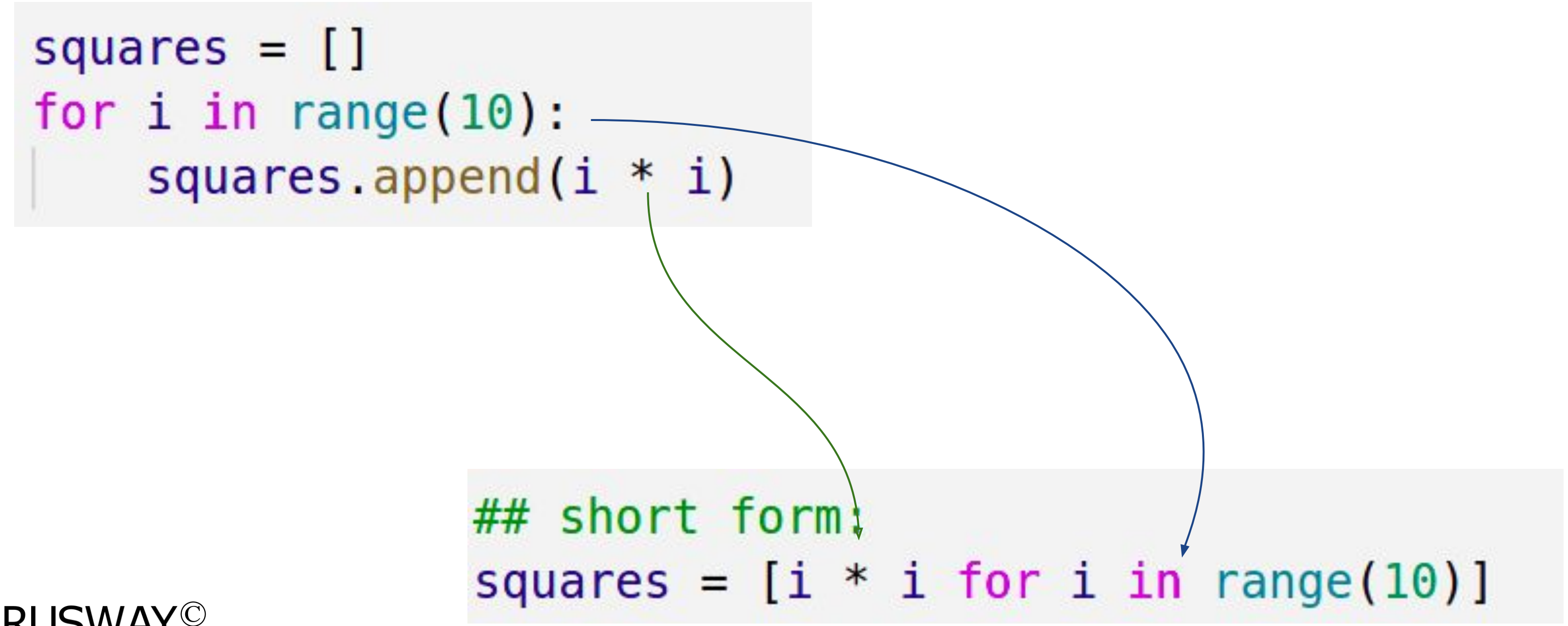
```
liste = []
for <var> in <iterable>:
          <expression>
```

short form (Pythonic way of coding: Less code – more effectiveness.);

```
liste = [<expression> for <var> in <iterable>]
```

List Comprehensions (List Generators)

<u>Written in a long form;</u>



```
squares = []
for i in range(10):
    squares.append(i * i)
```

```
## short form:
squares = [i * i for i in range(10)]
```

List Comprehensions (List Generators)

Written in a long form;

```
for <var> in <iterable>:
    if <condition>:
        <expression>
```

short form;

```
[ <expression> for <var> in <iterable> if <condition> ]
```

List Comprehensions (List Generators)

You create a list using a for loop and a **range()** function. (The following expression defines a generator for all the even numbers in 0-10):

```
evens = []
for n in range(12):
    if n % 2 == 0:
        evens.append(n)

print(evens)
```

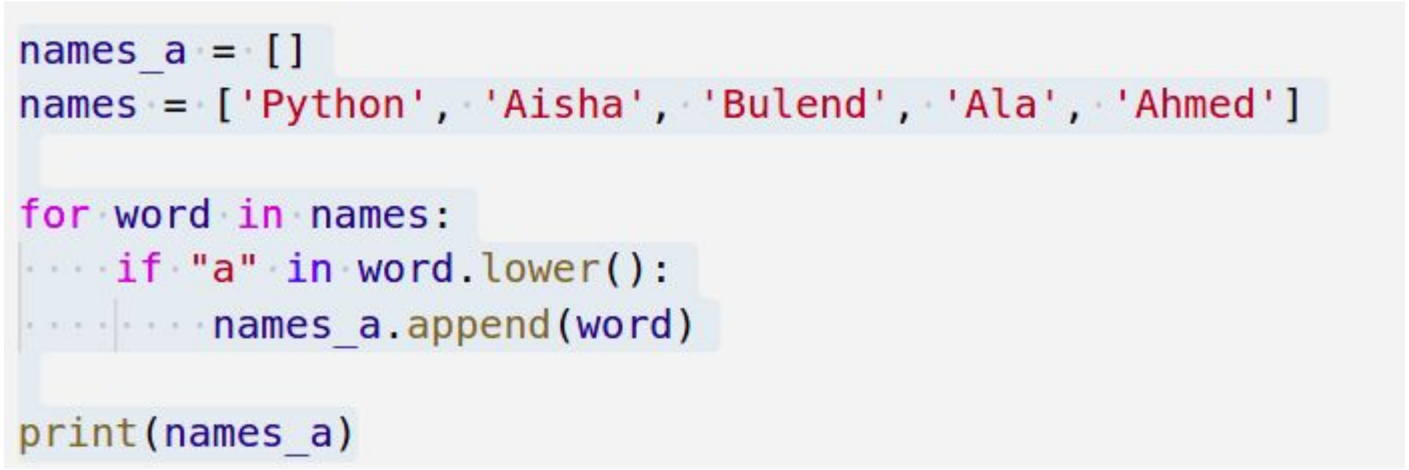
Pythonic way of coding: Less code – more effectiveness.

```
evens = [n for n in range(12) if n%2 == 0]
print(evens)
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

List Comprehensions (List Generators)

The following code stores words that contain the letter "a", in a list:



```
names_a = []
names = ['Python', 'Aisha', 'Bulend', 'Ala', 'Ahmed']

for word in names:
    if "a" in word.lower():
        names_a.append(word)

print(names_a)
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

List Comprehensions (List Generators)

```
names_a = []
names = ['Python', 'Aisha', 'Bulend', 'Ala', 'Ahmed']

for word in names:
    if "a" in word.lower():
        names_a.append(word)

print(names_a)
```

This can be written in a single line, using a list comprehension:

```
names = ['Python', 'Aisha', 'Bulend', 'Ala', 'Ahmed']
names_a = [word for word in names if "a" in word.lower()]
print(names_a)
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

List Comprehensions (List Generators)

Written in a long form;

```
for <var> in <iterable>:
    if <condition>:
        <expression-1>
    else:
        <expression-2>
```

```
[<expr1> if condition else <expr2> for <var> in iterable]
```

Short form;

List Comprehensions (List Generators)

Written in a long form;

```
even_or_sqr = []
for n in range(10):
    if n % 2 == 0:
        even_or_sqr.append(n)
    else:
        even_or_sqr.append(n**2)

print(even_or_sqr)
```

short form;

```
even_or_sqr = [n if n % 2 == 0 else n**2 for n in range(10)]
print(even_or_sqr)
```

List Comprehensions (List Generators)

#list comprehension samples:

```
[i**2 for i in range(10)]     #out-1: ?

sum([n for n in range(75)])     #out-2: ?

cumle = "Mistakes are for people"
set(i for i in cumle if not i in 'aeiou')  #out-3: ?
```