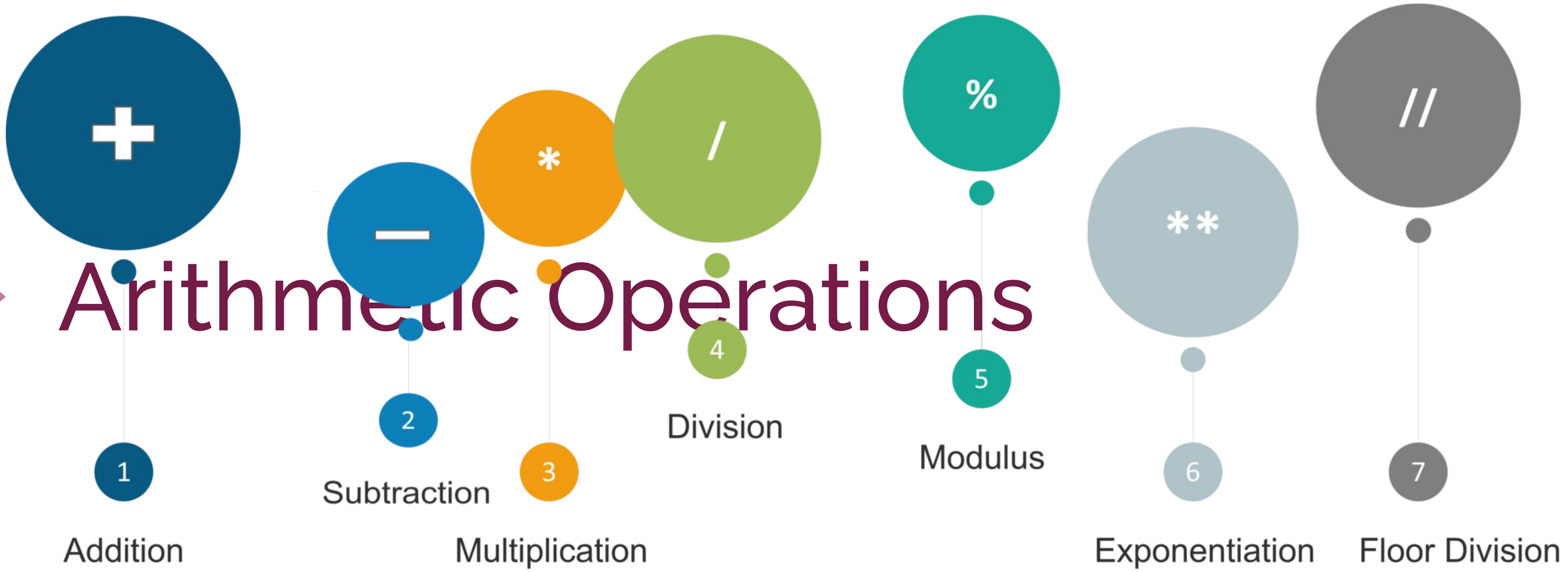
Simple Operations

CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents

Arithmetic Operations

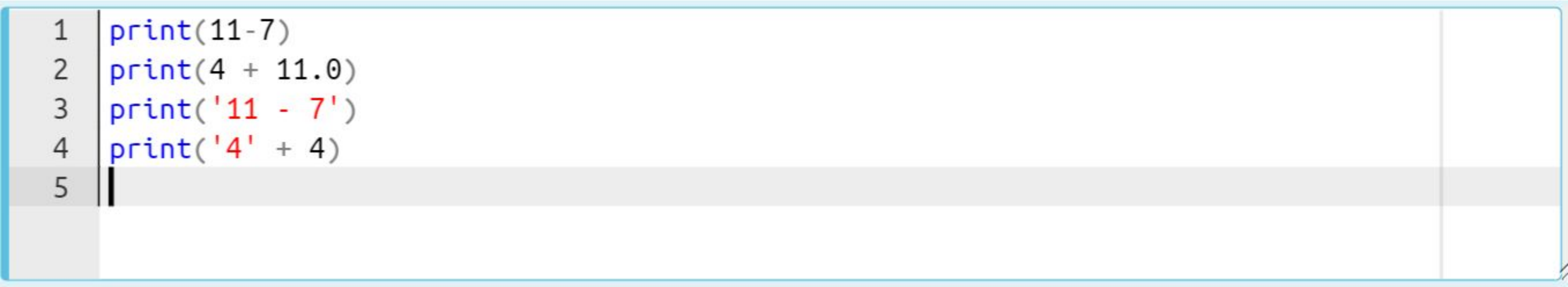


Arithmetic Operations

Operator	Description	Example
`+`	Addition operator	`100 + 45 = 145`
`-`	Subtraction operator	`500 - 65 = 435`
`*`	Multiplication operator	`25 * 4 = 100`
`/`	Float Division Operator	`10 / 2 = 5.0`
`//`	Integer Division Operator	`11 // 2 = 5`
`**`	Exponentiation Operator	`5 ** 3 = 125`
`%`	Remainder Operator	`10 % 3 = 1`

Arithmetic Operations

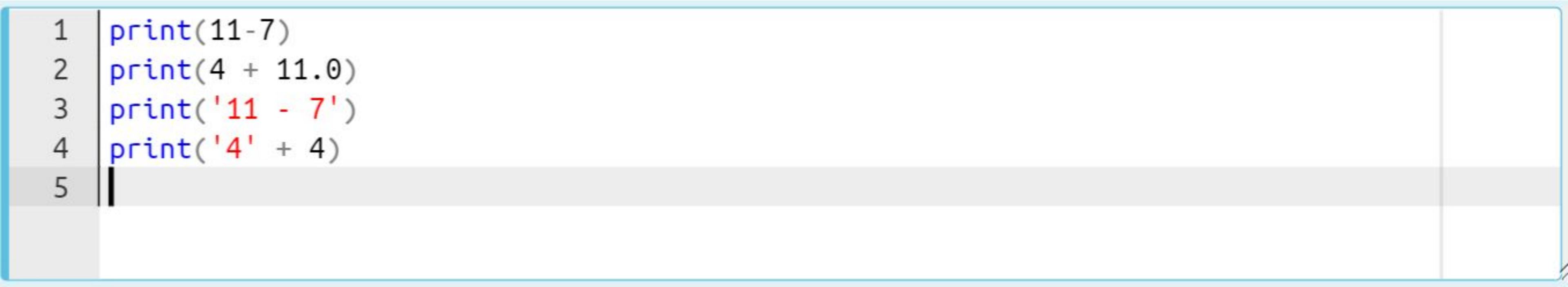
- Interactive question :



```
print(11-7)
print(4 + 11.0)
print('11 - 7')
print('4' + 4)

```

Arithmetic Operations

- The output :

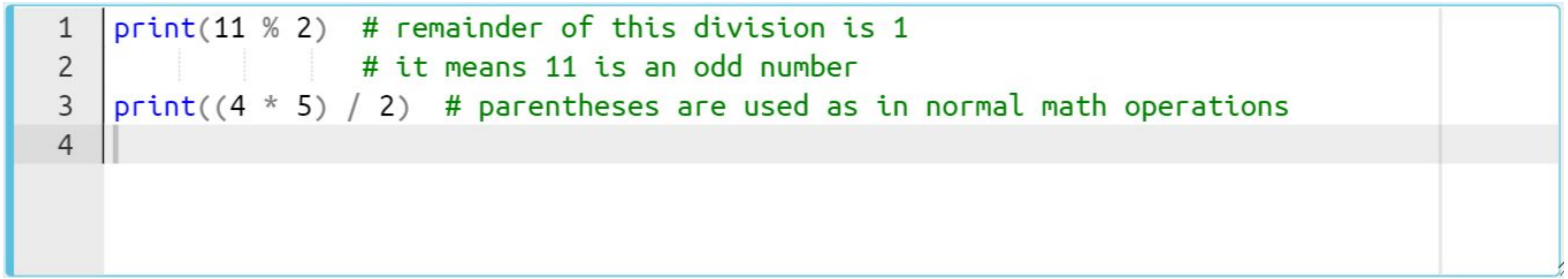


```python
print(11-7)
print(4 + 11.0)
print('11 - 7')
print('4' + 4)

```

```
4
15.0
11 - 7
Traceback (most recent call last):
  File "code.py", line 5, in <module>
    print('4'+ 4)
TypeError: can only concatenate str (not "int") to str
```

Arithmetic Operations

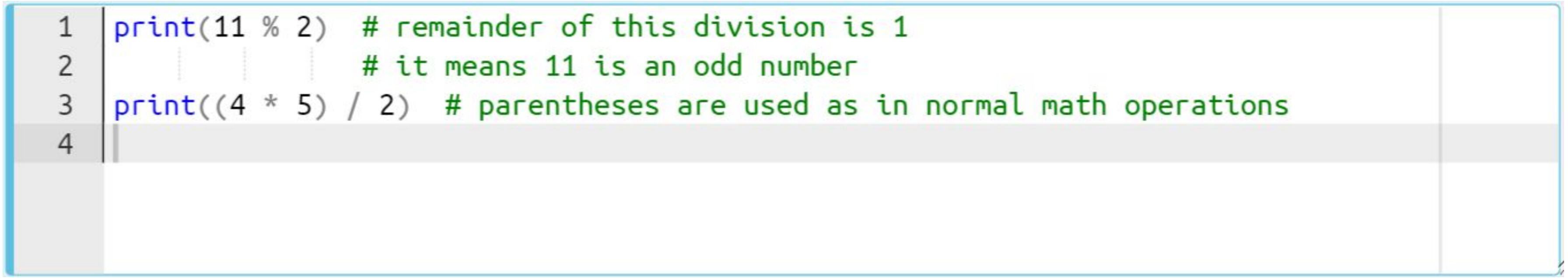
- Interactive question :



```
print(11 % 2)  # remainder of this division is 1
               # it means 11 is an odd number
print((4 * 5) / 2)  # parentheses are used as in normal math operations

```

Arithmetic Operations

- The output :



```
print(11 % 2)   # remainder of this division is 1
                # it means 11 is an odd number
print((4 * 5) / 2)  # parentheses are used as in normal math operations
```

```
1

10.0
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

Arithmetic Operations

💡Tips:

- `Variable` `math operator` `=` `number` gives the same result as `Variable` `=` `Variable` `math operator` `number`.
- `Variable` `+=` `number` gives the same result as `Variable = Variable + number`.

- `-=` decrements the variable in place,
- `+=` increment the variable in place,
- `*=` multiply the variable in place,
- `/=` divide the variable in place,
- `//=` floor divide the variable in place,
- `%=` returns the modulus of the variable in place,
- `**=` raise to power in place.

`x += 3` $\Longleftrightarrow$ `x = x + 3`

`x *= 3` $\Longleftrightarrow$ `x = x * 3`

`x **= 3` $\Longleftrightarrow$ `x = x ** 3`

Arithmetic Operations

1. parentheses : `()`
2. power : `**`
3. unary minus : `-3`
4. multiplication and division : `*`, `/`
5. addition and subtraction : `+`, `-`

Arithmetic Operations

- Interactive question :

```
a = (1 + 3 ) ** (2 ** (1 * 2 / 2) / 2)
print(a)
```

Arithmetic Operations

- The output :

```
a = (1 + 3 ) ** (2 ** (1 * 2 / 2) / 2)
print(a)
```

```
4.0
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

\n

\t

Escape Sequences

\b

Escape Sequences (review)

- Python ignores any character which comes immediately after `\`.

- `\n` : means new line,
- `\t` : means `tab` mark,
- `\b` : means backspace. It moves the cursor one character to the left.

Escape Sequences, Quiz

- **Let's** take a closer look at the escape sequences through the examples.

```
print('we are', '\boosting', 'our', '\brotherhood')
print('it\'s essential to learn Python\'s libraries in IT World')
print('C:\\north pole\noise_penguins.txt')
print('first', 'second', 'third', sep='\t')
```

Escape Sequences, Quiz

- **Let’s** take a closer look at the escape sequences through the examples.

```
print('we are', '\boosting', 'our', '\brotherhood')
print('it\'s essential to learn Python\'s libraries in IT World')
print('C:\\north pole\noise_penguins.txt')
print('first', 'second', 'third', sep='\t')
```

```
we areoosting ourrotherhood
it's essential to learn Python's libraries in IT World
C:\north pole
oise_penguins.txtfirst   second   third
```

Boolean Operations

CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents

Boolean Logic Expressions

not

and

or

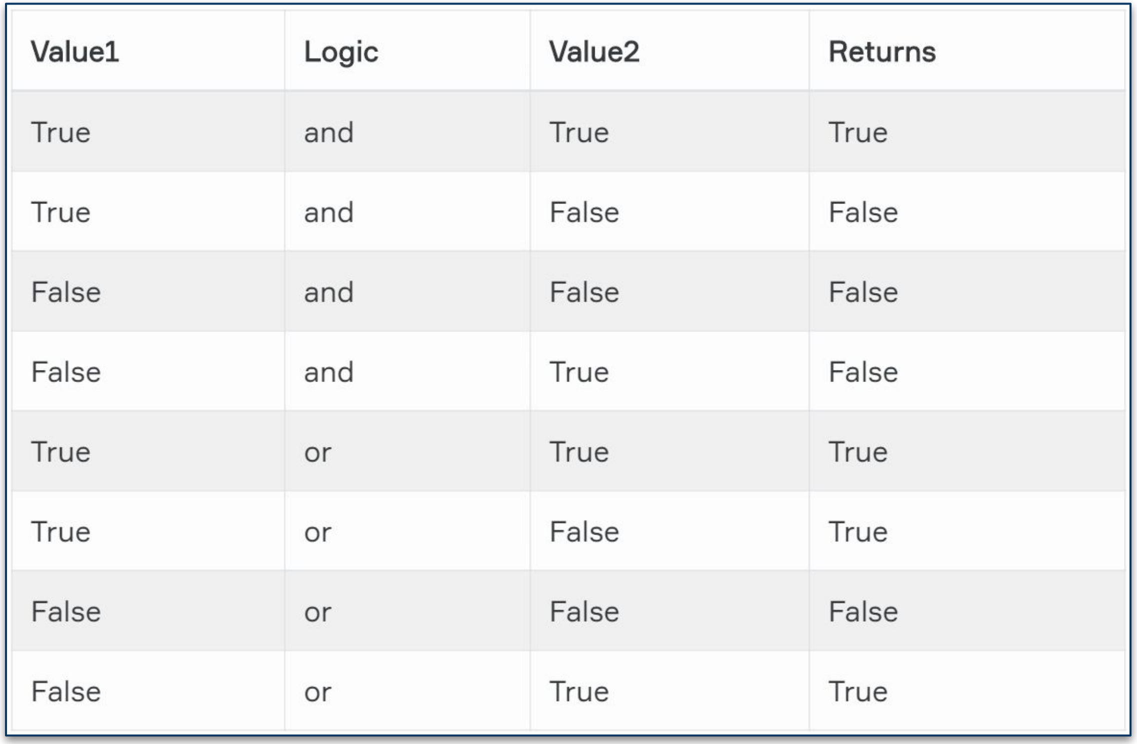
Boolean Logic Expressions

- There are three built-in operators in Python :

`and`	It evaluates all expressions and returns the **last** expression if **all** expressions are evaluated `True`. Otherwise, it returns the **first** value that evaluated `False`.
`or`	It evaluates the expressions left to right and returns the first value that evaluated `True` or the last value (if none is `True`).
`not`	It evaluates the expression that follows it as the opposite of the truth. eg. `not True` means `False`

Boolean Logic Expressions

- Table of Logic Expressions in Python :



Value1	Logic	Value2	Returns
True	and	True	True
True	and	False	False
False	and	False	False
False	and	True	False
True	or	True	True
True	or	False	True
False	or	False	False
False	or	True	True

It's better to keep this table in mind.

Order of Priority

Order of Priority

- Here are the operators **in order** of their **priorities** :

1. `not`
2. `and`
3. `or`

Order of Priority

- It is important to remember that, logical operators have a different priority and it has an effect on the order of evaluation.
- Here are the operators in order of their priorities :

1. `not`
2. `and`
3. `or`

```
bool_var = False and not True
print(bool_var)
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

Order of Priority

- It is important to remember that, logical operators have a different priority and it has an effect on the order of evaluation.
- Here are the operators in order of their p

Firstly evaluated.
The result = `False`

1. `not`
2. `and`
3. `or`

```
bool_var = False and not True
print(bool_var)
```

Order of Priority

- It is important to remember that, logical operators have a different priority and it has an effect on the order of evaluation
- Here are the op er of their p

1. `not`
2. `and`
3. `or`

```python
bool_var = False and not True
print(bool_var)
```

Secondly evaluated.
`False and False = False`

Firstly evaluated.
The result = `False`

CLARUSWAY©
WAY TO REINVENT YOURSELF

Order of Priority

- It is important to remember that, logical operators have a different priority and it has an effect on the order of evaluation.
- Here are the op er of their p

1. not
2. and
3. or

Secondly evaluated.
`False and False = False`

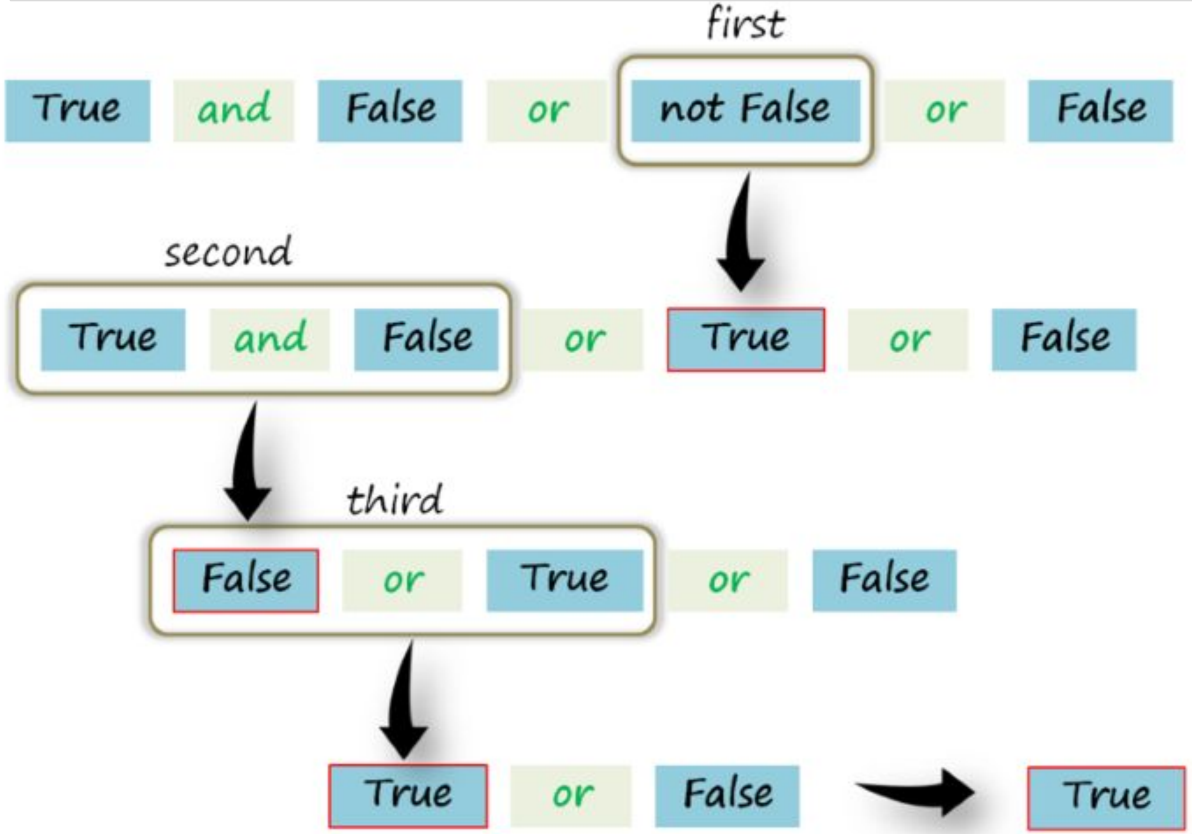
Firstly evaluated.
The result = `False`

```python
bool_var = False and not True
print(bool_var)
```

```
False
```

Order of Priority (review)

```
True and False or not False or False = ?
```



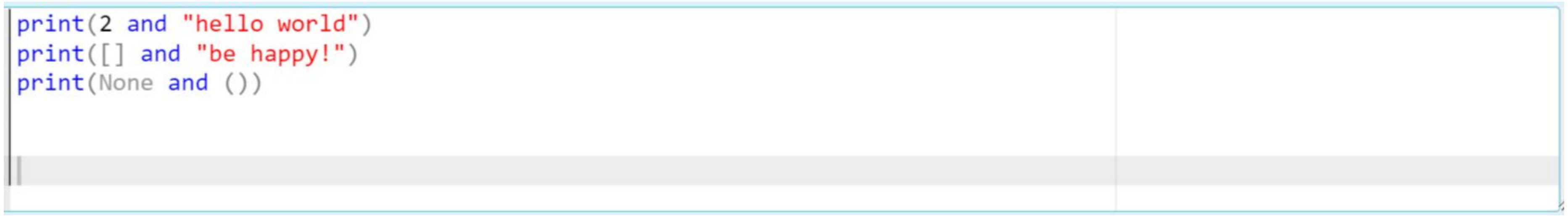
Truth Values of Logic Statements

Truth Values of Logic Statements

- **Falsy** values in Python:

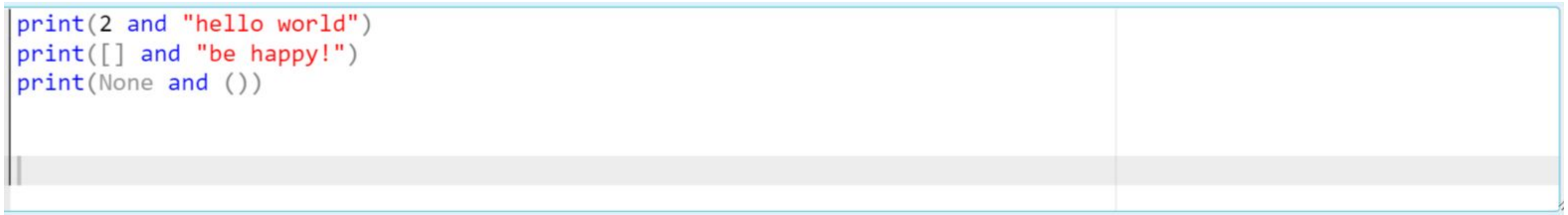
  - None
  - Zero : `0, 0.0, 0j`
  - Empty Seq. and collections : `'', [], {}`
  - Any remaining value : `True`

Truth Values of Logic Statements

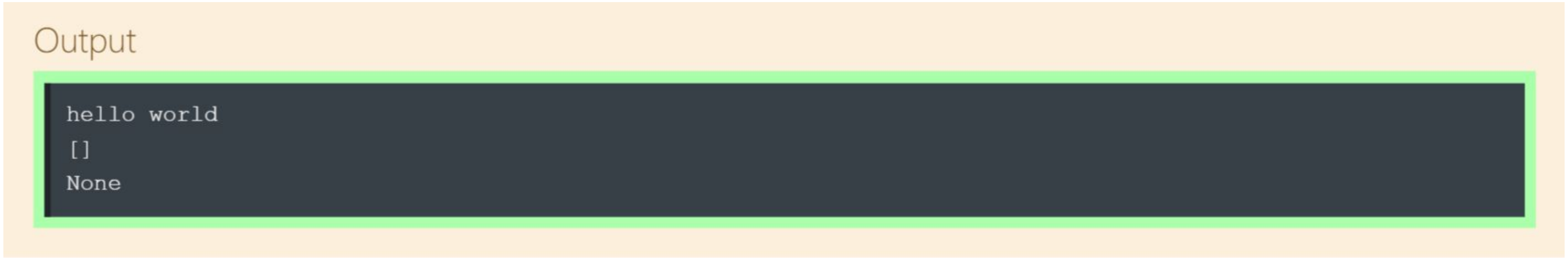


```
print(2 and "hello world")
print([] and "be happy!")
print(None and ())
```

Truth Values of Logic Statements



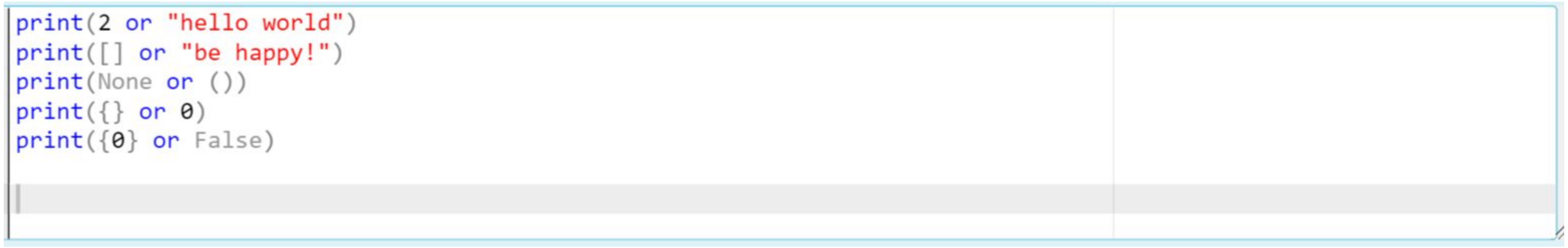
```python
print(2 and "hello world")
print([] and "be happy!")
print(None and ())
```



Output

```
hello world
[]
None
```

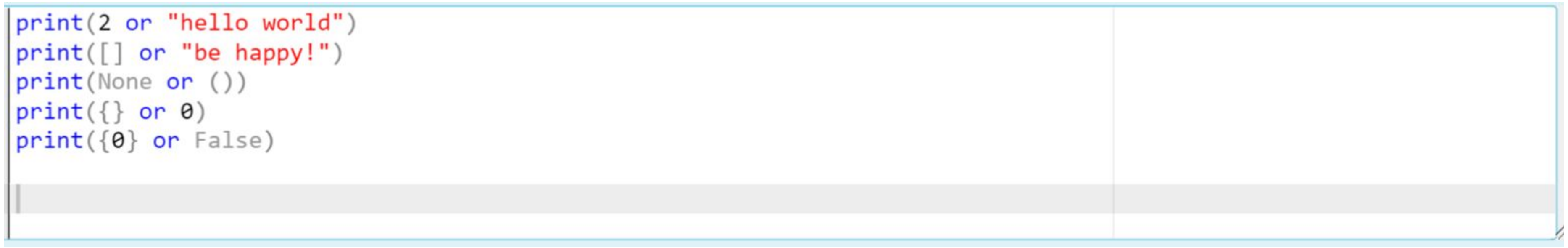
Truth Values of Logic Statements



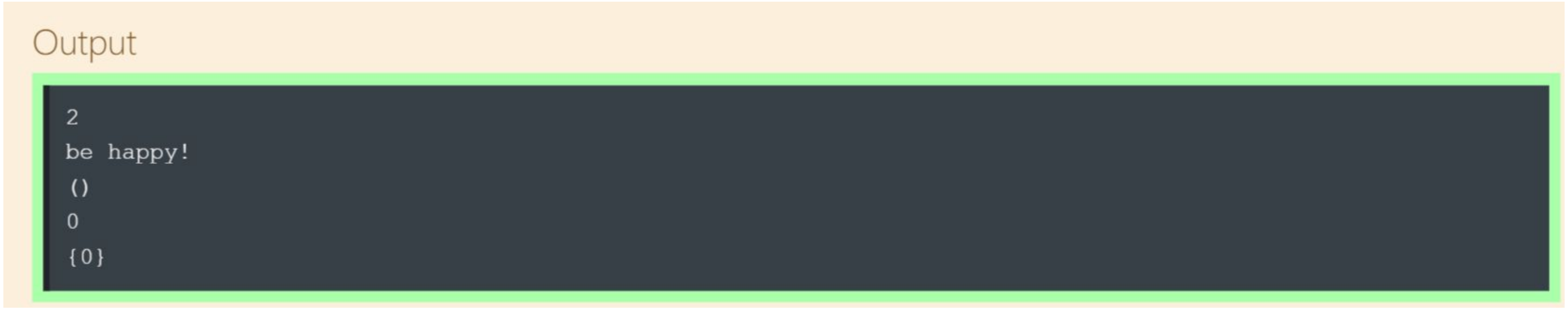
```
print(2 or "hello world")
print([] or "be happy!")
print(None or ())
print({} or 0)
print({0} or False)
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

Truth Values of Logic Statements



```
print(2 or "hello world")
print([] or "be happy!")
print(None or ())
print({} or 0)
print({0} or False)
```



Output

```
2
be happy!
()
0
{0}
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

Indexing & Slicing Strings

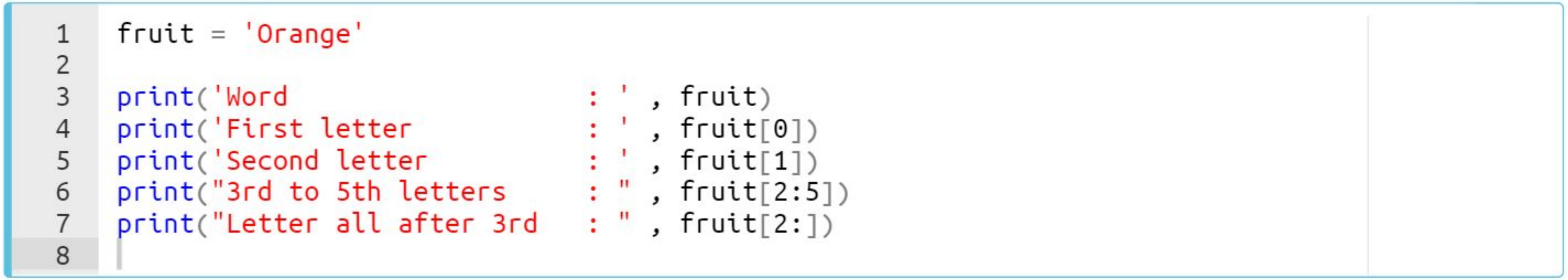
```
best ='Clarusway'
```

```
best[2]
```

```
best[2:]
```

Indexing&Slicing Strings

- Let's elaborate on this example :

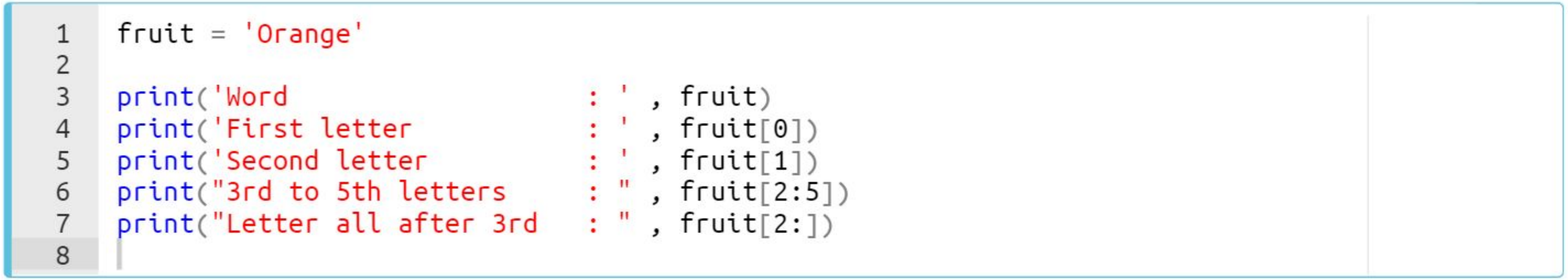


```
fruit = 'Orange'

print('Word                    : ' , fruit)
print('First letter            : ' , fruit[0])
print('Second letter           : ' , fruit[1])
print("3rd to 5th letters      : " , fruit[2:5])
print("Letter all after 3rd    : " , fruit[2:])

```

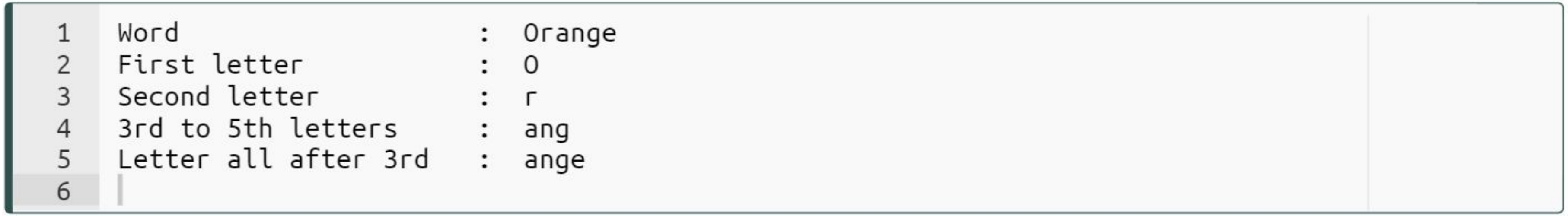
CLARUSWAY©
WAY TO REINVENT YOURSELF

Indexing&Slicing Strings

- Let's elaborate on this example :



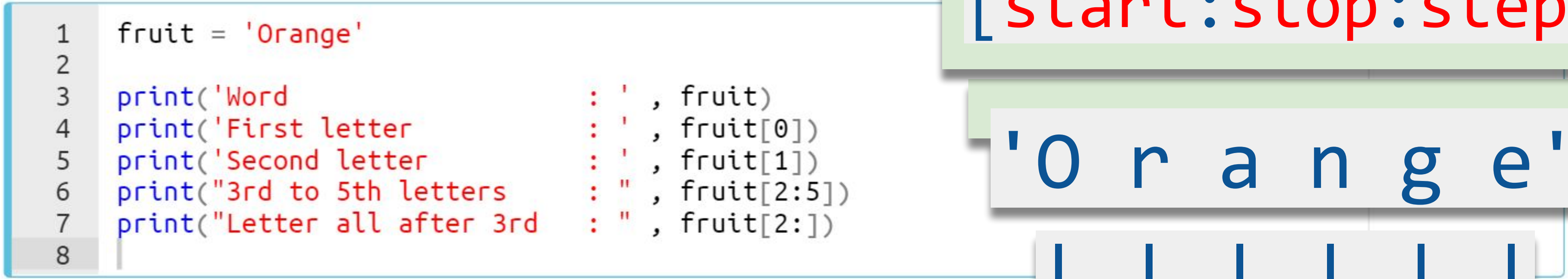
```python
fruit = 'Orange'

print('Word                 : ' , fruit)
print('First letter         : ' , fruit[0])
print('Second letter        : ' , fruit[1])
print("3rd to 5th letters   : " , fruit[2:5])
print("Letter all after 3rd : " , fruit[2:])

```



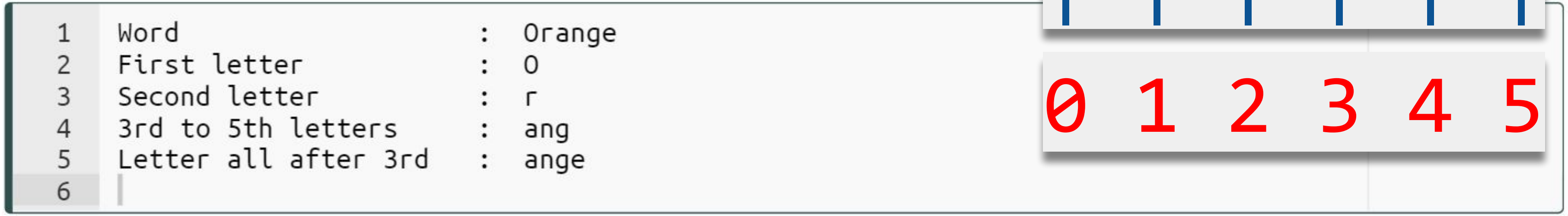
```
Word                 :  Orange
First letter         :  O
Second letter        :  r
3rd to 5th letters   :  ang
Letter all after 3rd :  ange

```

Indexing&Slicing Strings

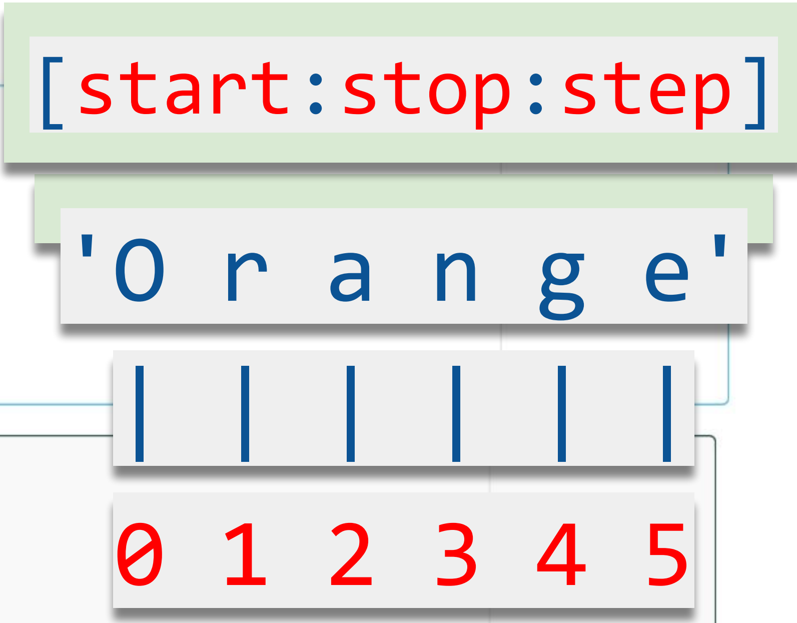
- Let's elaborate on this example :



```python
fruit = 'Orange'

print('Word                 : ' , fruit)
print('First letter         : ' , fruit[0])
print('Second letter        : ' , fruit[1])
print("3rd to 5th letters   : " , fruit[2:5])
print("Letter all after 3rd : " , fruit[2:])
```

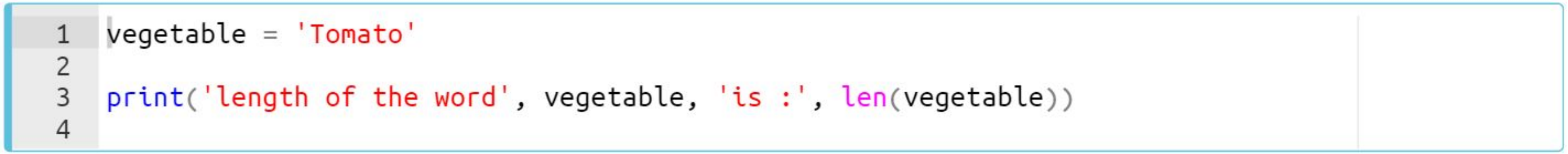


```
Word                 :  Orange
First letter         :  O
Second letter        :  r
3rd to 5th letters   :  ang
Letter all after 3rd :  ange
```



Indexing&Slicing Strings

- `len()` function measure the length of any iterable :

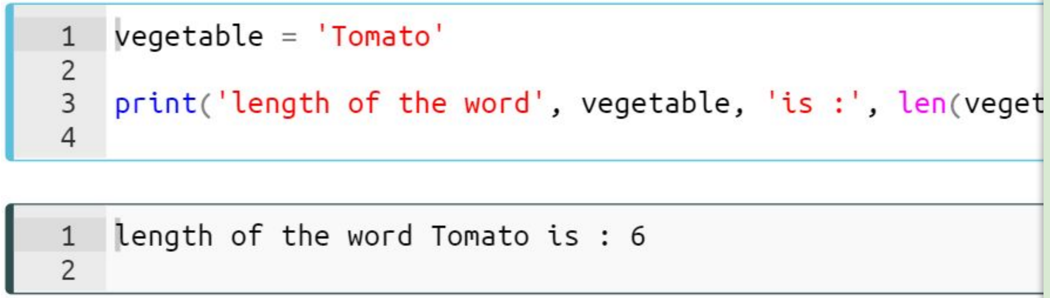


```
vegetable = 'Tomato'

print('length of the word', vegetable, 'is :', len(vegetable))

```

CLARUSWAY©
WAY TO REINVENT YOURSELF

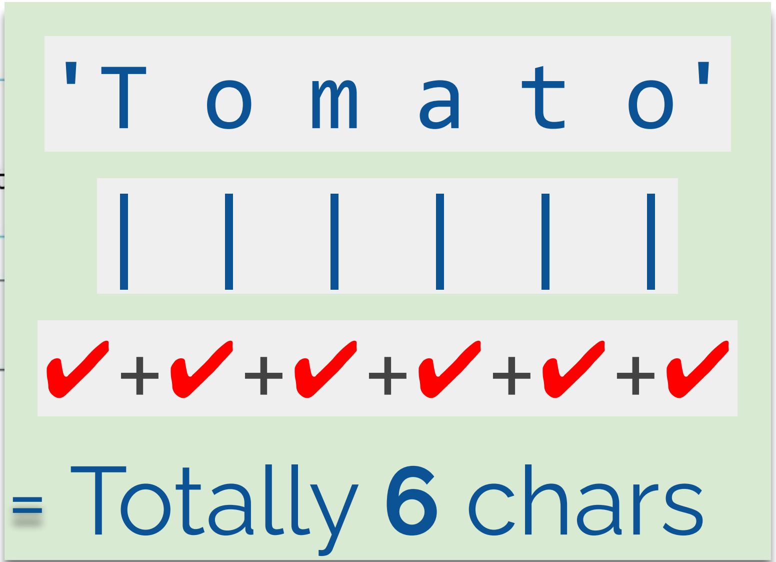
Indexing&Slicing Strings

- The output :



```
vegetable = 'Tomato'

print('length of the word', vegetable, 'is :', len(veget
```

```
length of the word Tomato is : 6
```



String Formatting

String Formatting with Arithmetic Syntax

String Formatting with Arithmetic Syntax

- Here are basic operators :

- `+`
- `=`
- `*`

String Formatting with Arithmetic Syntax

- We can use arithmetic operator syntaxes in string formatting operations
- Here are basic operators :

- +
- =
- *

```
str_one = 'upper'
str_two = 'case'
str_comb = str_one + str_two
print('upper' + 'case')
print(str_one + str_two)
print(str_comb)
```

String Formatting with Arithmetic Syntax

- We can use arithmetic operator syntaxes in string formatting operations
- Here are basic operators :

- `+`
- `=`
- `*`

```
str_one = 'upper'
str_two = 'case'
str_comb = str_one + str_two
print('upper' + 'case')
print(str_one + str_two)
print(str_comb)
```

```
uppercase
uppercase
uppercase
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

String Formatting with Arithmetic Syntax

- Another example :

- +
- =
- *

```
str_one = 'upper'
str_two = 3 * 'upper'
str_comb = str_one * 3
print(str_two)
print(str_comb)
print(* str_one)
```

String Formatting with Arithmetic Syntax

- Another example :

- +
- =
- *

```
str_one = 'upper'
str_two = 3 * 'upper'
str_comb = str_one * 3
print(str_two)
print(str_comb)
print(* str_one)
```

```
upperupperupper
upperupperupper
u p p e r
```

String Formatting with Arithmetic Syntax

- Another example :

- +
- =
- *

```
str_one = 'upper'
str_two = 3 * 'upper'
str_comb = str_one * 3
print(str_two)
print(str_comb)
print(* str_one)
```

Separates the string into its elements

```
upperupperupper
upperupperupper
u p p e r
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

String Formatting with Arithmetic Syntax

- Another example :

- ● +
- ● =
- ● *

```
str_one = 'upper'
str_one += 'case'
print(str_one)
str_one += 'letter'
print(str_one)
str_one += 'end'
print(str_one)
```

String Formatting with Arithmetic Syntax

- Another example :

- +
- =
- *

```
str_one = 'upper'
str_one += 'case'
print(str_one)
str_one += 'letter'
print(str_one)
str_one += 'end'
print(str_one)
```

```
uppercase
uppercaseletter
uppercaseletterend
```

```
str1 = str1 + str
str1 += str
```

```
str1 = str1 * 2
str1 *= 2
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

String Formatting

String Formatting with `string.format()` Method

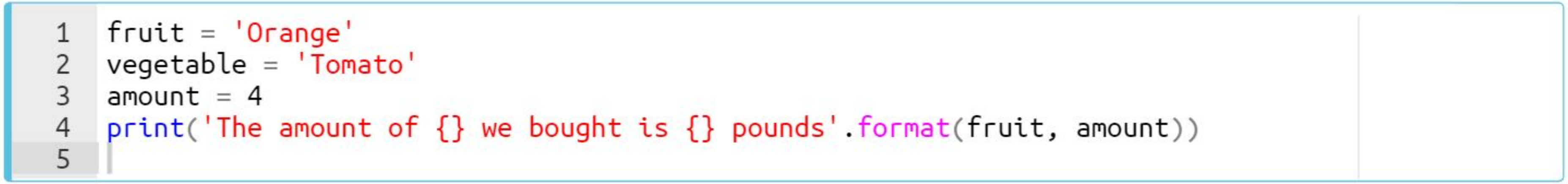
String Formatting with `string.format()` Method

- The formula syntax 👇

```
'string {} string {} string'.format(data1, data2)
```

String Formatting with `string.format()` Method

- Take a look at the example 👇



```python
fruit = 'Orange'
vegetable = 'Tomato'
amount = 4
print('The amount of {} we bought is {} pounds'.format(fruit, amount))
```

String Formatting

String Formatting with `f-string`

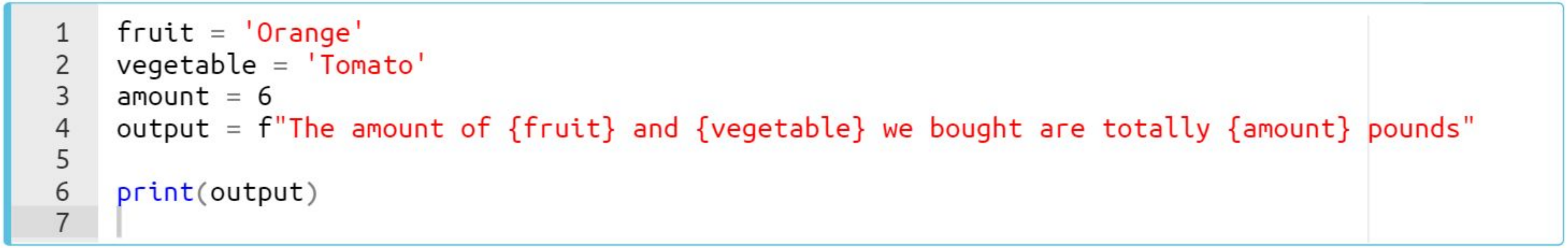
String Formatting with `f-string`

- The formula syntax 👇

```
f'string {variable1} string {variable2} string'
```

String Formatting with `f-string`

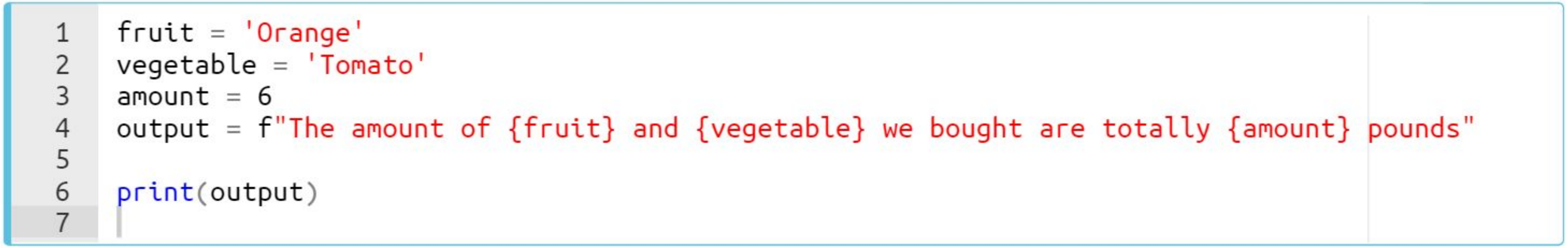
- Take a look at the example 👇



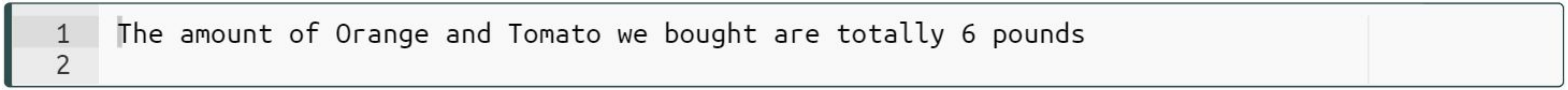
```python
fruit = 'Orange'
vegetable = 'Tomato'
amount = 6
output = f"The amount of {fruit} and {vegetable} we bought are totally {amount} pounds"

print(output)
```

String Formatting with `f-string`

- Take a look at the example 👇



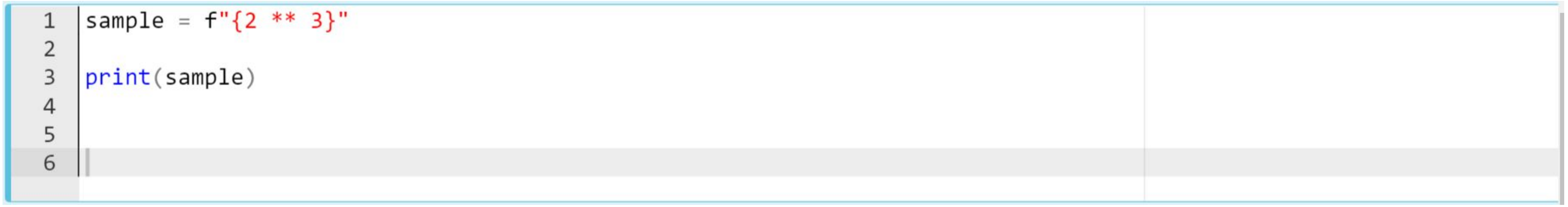
```python
fruit = 'Orange'
vegetable = 'Tomato'
amount = 6
output = f"The amount of {fruit} and {vegetable} we bought are totally {amount} pounds"

print(output)

```



```
The amount of Orange and Tomato we bought are totally 6 pounds

```

String Formatting with `f-string`

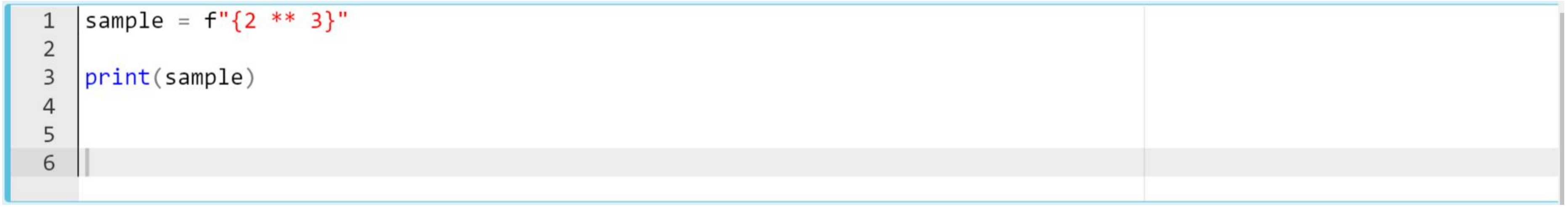
- You can use all valid expressions, variables, and even methods in curly braces. 👇



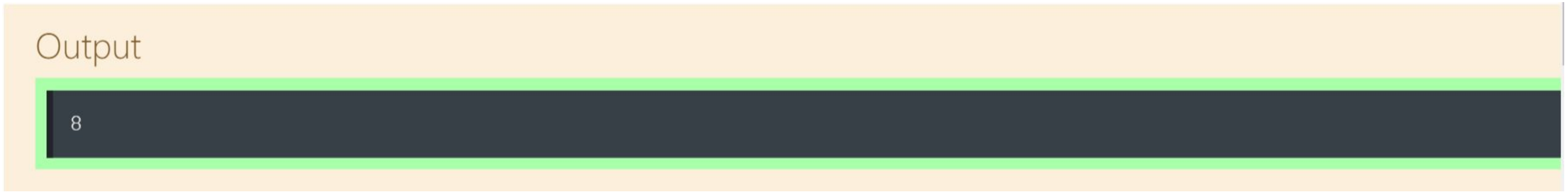
```python
sample = f"{2 ** 3}"

print(sample)


```

String Formatting with `f-string`

- You can use all valid expressions, variables, and even methods in curly braces. 👇



```python
sample = f"{2 ** 3}"

print(sample)
```



Output

```
8
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

String Formatting with `f-string`

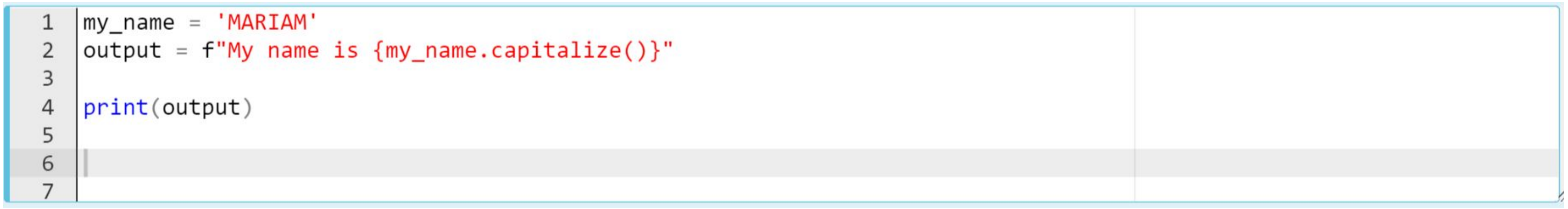
- **Task :**
  - Type a Python code to get the output of "`My name is Mariam`", using `.capitalize()` and `f-string` methods with the `name` variable below.

```
name = "MARIAM"
```

You're familiar with `.capitalize()` method from **pre-class** materials

String Formatting with `f-string`

- The code should be like :



```python
my_name = 'MARIAM'
output = f"My name is {my_name.capitalize()}"

print(output)


```

CLARUSWAY©
WAY TO REINVENT YOURSELF

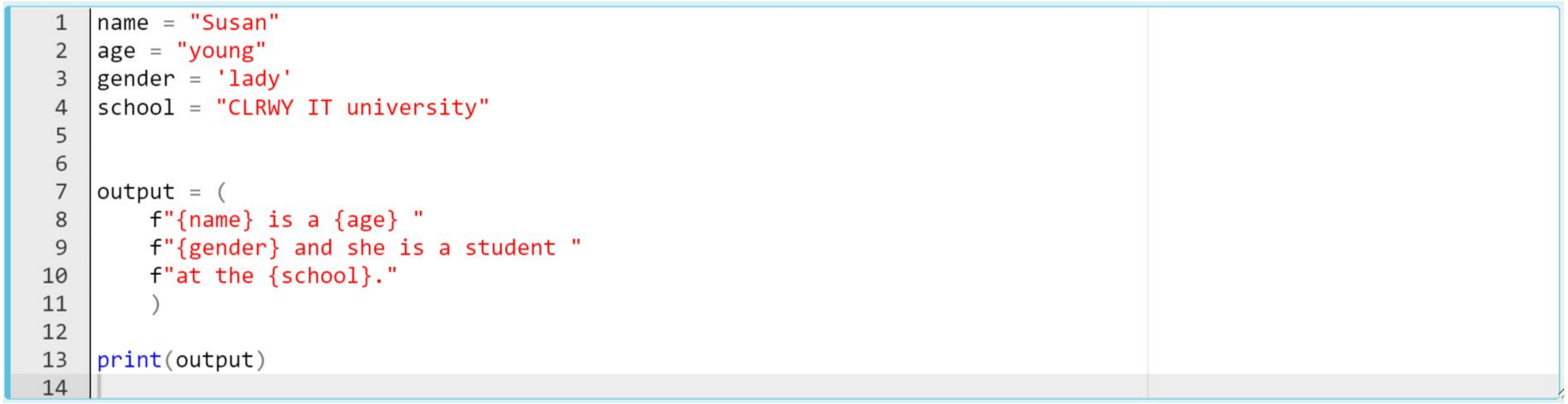
String Formatting with `f-string`

- **Task :**
  - Type a Python code to get the output of "`Susan is a young lady and she is a student at the CLRWY IT university.`", using `f-string` in *multiline* with the `variables` below.

```
name = "Susan"
age = "young"
gender = "lady"
school = "CLRWY IT university"
```

String Formatting with `f-string`

- The code should be like :



```python
name = "Susan"
age = "young"
gender = 'lady'
school = "CLRWY IT university"


output = (
    f"{name} is a {age} "
    f"{gender} and she is a student "
    f"at the {school}."
    )

print(output)

```