

Watch by Tuesday, October 13, 2020 | **Lesson #3**

# Lists, object types, and logical operations

OCEAN 215 | Autumn 2020

Ethan Campbell and **Katy Christensen**

# What we'll cover in this lesson

---

1. What is a list?
2. List functions
3. Object types
4. Logical Operations

# What we'll cover in this lesson

---

- 1. What is a list?**
2. List functions
3. Object types
4. Logical Operations

# Lists are objects with length

---

**Remember: strings have a length that includes each character**

```
1 str_ex = 'This is an example string'  
2 str_ex_len = len(str_ex)  
3 print(str_ex, '(', str_ex_len, 'characters )')
```

```
↳ This is an example string ( 25 characters )
```

**Lists also have length! The length of a list includes all of the items within...**

# A list can have any object type as an item

---

**Remember: strings have a length that includes each character**

```
1 str_ex = 'This is an example string'
2 str_ex_len = len(str_ex)
3 print(str_ex, '(', str_ex_len, 'characters )')
```

☞ This is an example string ( 25 characters )

**Lists also have length! The length of a list includes all of the items within...**

**A list can have numbers.**

```
1 listnum_ex = [1, 2.45, 3e10, 4, -5]
2 listnum_len = len(listnum_ex)
3 print(listnum_ex, '(', listnum_len, 'items )')
```

☞ [1, 2.45, 30000000000.0, 4, -5] ( 5 items )

# A list can have any object type as an item

---

**Remember: strings have a length that includes each character**

```
1 str_ex = 'This is an example string'
2 str_ex_len = len(str_ex)
3 print(str_ex, '(', str_ex_len, 'characters )')
```

☞ This is an example string ( 25 characters )

**Lists also have length! The length of a list includes all of the items within...**

**A list can have numbers.**

```
1 listnum_ex = [1, 2.45, 3e10, 4, -5]
2 listnum_len = len(listnum_ex)
3 print(listnum_ex, '(', listnum_len, 'items )')
```

☞ [1, 2.45, 30000000000.0, 4, -5] ( 5 items )

**To put objects into a list, use square brackets [ ]**

# A list can have any object type as an item

---

**Remember: strings have a length that includes each character**

```
1 str_ex = 'This is an example string'
2 str_ex_len = len(str_ex)
3 print(str_ex, '(', str_ex_len, 'characters )')
```

☞ This is an example string ( 25 characters )

**Lists also have length! The length of a list includes all of the items within...**

**A list can have numbers.**

```
1 listnum_ex = [1, 2.45, 3e10, 4, -5]
2 listnum_len = len(listnum_ex)
3 print(listnum_ex, '(', listnum_len, 'items )')
```

☞ [1, 2.45, 30000000000.0, 4, -5] ( 5 items )

**A list can have Booleans.**

```
1 listbool_ex = [True, False, False, True, False]
2 listbool_len = len(listbool_ex)
3 print(listbool_ex, '(', listbool_len, 'items )')
```

☞ [True, False, False, True, False] ( 5 items )

# A list can have any object type as an item

---

**Remember: strings have a length that includes each character**

```
1 str_ex = 'This is an example string'
2 str_ex_len = len(str_ex)
3 print(str_ex, '(', str_ex_len, 'characters )')
```

↳ This is an example string ( 25 characters )

**Lists also have length! The length of a list includes all of the items within...**

**A list can have numbers.**

```
1 listnum_ex = [1, 2.45, 3e10, 4, -5]
2 listnum_len = len(listnum_ex)
3 print(listnum_ex, '(', listnum_len, 'items )')
```

↳ [1, 2.45, 30000000000.0, 4, -5] ( 5 items )

**A list can have Booleans.**

```
1 listbool_ex = [True, False, False, True, False]
2 listbool_len = len(listbool_ex)
3 print(listbool_ex, '(', listbool_len, 'items )')
```

↳ [True, False, False, True, False] ( 5 items )

**A list can have strings.**

```
1 liststr_ex = ['This', 'is', 'an', 'example', 'list']
2 liststr_len = len(liststr_ex)
3 print(liststr_ex, '(', liststr_len, 'items )')
```

↳ ['This', 'is', 'an', 'example', 'list'] ( 5 items )



# A list can have any object type as an item

## A list can have numbers.

```
1 listnum_ex = [1, 2.45, 3e10, 4, -5]
2 listnum_len = len(listnum_ex)
3 print(listnum_ex, '(', listnum_len, 'items )')
```

```
↳ [1, 2.45, 30000000000.0, 4, -5] ( 5 items )
```

## A list can have Booleans.

```
1 listbool_ex = [True, False, False, True, False]
2 listbool_len = len(listbool_ex)
3 print(listbool_ex, '(', listbool_len, 'items )')
```

```
↳ [True, False, False, True, False] ( 5 items )
```

## A list can have strings.

```
1 liststr_ex = ['This', 'is', 'an', 'example', 'list']
2 liststr_len = len(liststr_ex)
3 print(liststr_ex, '(', liststr_len, 'items )')
```

```
↳ ['This', 'is', 'an', 'example', 'list'] ( 5 items )
```

## A list can have lists.

```
1 listlist_ex = [['This', 'is', 'an', 'example', 'list'],
                 [True, False, False, True, False],
                 [1, 2.45, 3e10, 4, -5]]
2
3
4
5 listlist_len = len(listlist_ex)
6 print(listlist_ex, '(', listlist_len, 'items )')
```

```
↳ [['This', 'is', 'an', 'example', 'list'], [True, False, False, True, False], [1, 2.45, 30000000000.0, 4, -5]] ( 3 items )
```

**Another way to create this list  
would be to use the variable names  
for each of the previously created  
lists!**

```
1 listlist_ex = [liststr_ex,
                 listbool_ex,
                 listnum_ex]
2
3
4
```

# A list can have any object type as an item

---

## A list can have a mix of object types.

```
1 # Book information: Title, Author, Year(s) Published, Pages, Available in Library
2 book_info = ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
3
4 print(book_info)
```

```
↳ ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
```

## A list can also be empty.

```
1 listemp_ex = []
2 listemp_len = len(listemp_ex)
3 print(listemp_ex, '(' , listemp_len, 'items ')')
```

```
↳ [] ( 0 items )
```

# List indexing and slicing

## Remember:

String indexing

P	y	t	h	o	n		i	s		f	u	n	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13

How python counts list items (indexing):

```
1 # Book information: Title, Author, Year(s) Published, Pages, Available in Library
2 book_info = ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
3
4 print(book_info)
```

↳ ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]

0

1

2

3

4

**Indexing and slicing is the same for lists and strings**

Single items	Slices
<code>book_info[1]</code>	<code>book_info[0:2]</code>
<code>'Miguel de Cervantes'</code>	<code>['Don Quixote', 'Miguel de Cervantes']</code>

# List indexing and slicing

## Remember:

String indexing

P	y	t	h	o	n		i	s		f	u	n	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13

How python counts list items (indexing):

```
1 # Book information: Title, Author, Year(s) Published, Pages, Available in Library
2 book_info = ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
3
4 print(book_info)
```

↳ ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]

0

1

2

3

4

## Multi-level indexing

book\_info[1]

'Miguel de Cervantes'

book\_info[1][0:6]

'Miguel'

**List items that are objects with length can be sliced**

# List indexing and slicing

## Remember:

String indexing

P	y	t	h	o	n		i	s		f	u	n	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13

How python counts list items (indexing):

```
1 # Book information: Title, Author, Year(s) Published, Pages, Available in Library
2 book_info = ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
3
4 print(book_info)
```

↳ ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]

0	1	2	3	4
-5	-4	-3	-2	-1

## Negative indexing

book\_info[-1]

book\_info[4]

True

book\_info[-3]

book\_info[2]

[1605, 1615]

# List indexing and slicing

```
1 # Book information: Title, Author, Year(s) Published, Pages, Available in Library
2 book_info = ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
3
4 print(book_info)
```

```
☞ ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
```

<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>-5</b>	<b>-4</b>	<b>-3</b>	<b>-2</b>	<b>-1</b>

Items in a list can be replaced using their index values

```
1 # Book information: Title, Author, Year(s) Published, Pages, Available in Library
2 book_info = ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
3 print(book_info)
4
5 book_info[-1] = False
6 print(book_info)
```

**This could also be written as  
book\_info [ 4 ] = False**

```
☞ ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
   ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, False]
```

# List indexing and slicing

```
1 # Book information: Title, Author, Year(s) Published, Pages, Available in Library
2 book_info = ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
3
4 print(book_info)
```

```
☞ ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
```

<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>-5</b>	<b>-4</b>	<b>-3</b>	<b>-2</b>	<b>-1</b>

**If you are starting at the beginning or stopping at the end of a list, you can omit the index value in your slice**

## More Slicing

`book_info[:3]`

`book_info[0:3]`

`['Don Quixote', 'Miguel de Cervantes', [1605,1615]]`

`book_info[2:]`

`book_info[2:5]`

`[[1605, 1615], 863, True]`

# List indexing and slicing

```
1 # Book information: Title, Author, Year(s) Published, Pages, Available in Library
2 book_info = ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
3
4 print(book_info)
```

```
☞ ['Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True]
```

<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>-5</b>	<b>-4</b>	<b>-3</b>	<b>-2</b>	<b>-1</b>

**You can slice using a step argument to get every nth (1st, 2nd, 3rd, 4th, etc.) items in a list**

Extended Slicing		
<code>book_info[0:5:1]</code>	<code>book_info[::1]</code>	<code>['Don Quixote', 'Miguel de Cervantes', [1605,1615],863, True]</code>
<code>book_info[0:5:2]</code>	<code>book_info[::2]</code>	<code>['Don Quixote', [1605,1615], True]</code>
<code>book_info[0:5:3]</code>	<code>book_info[::3]</code>	<code>['Don Quixote', 863]</code>
<code>book_info[0:5:-1]</code>	<code>book_info[::-1]</code>	<code>[True, 863, [1605,1615], 'Miguel de Cervantes', 'Don Quixote']</code>



# Objects like lists...

## Tuple

A tuple is the same as a list except they are immutable - we cannot change the items inside once they are assigned. Create a tuple using parentheses ( ) around the objects you want inside.

```
▶ 1 book_info = ('Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True)
   2 print(book_info)
   3
   4 book_info[-1] = False
   5 print(book_info)
```

```
↳ ('Don Quixote', 'Miguel de Cervantes', [1605, 1615], 863, True)
```

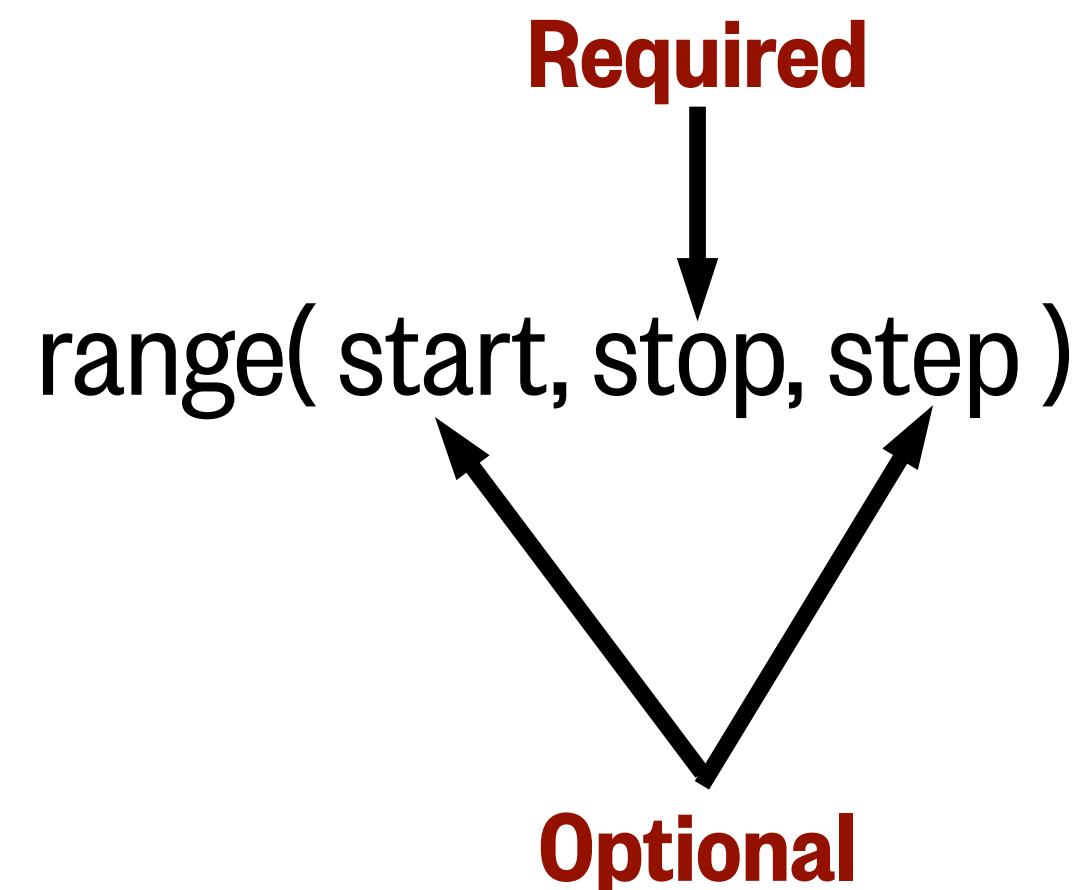
```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-49-9a102551762e> in <module>()
      2 print(book_info)
      3
----> 4 book_info[-1] = False
      5 print(book_info)
```

```
TypeError: 'tuple' object does not support item assignment
```

# Objects like lists...

## Range

A range object creates a sequence of numbers. The sequence starts from zero and increases by ones by default. The object created is a range object, but specific numbers in the sequence can be called using the same indexing as lists.



```
1 rn1 = range(10)
2 print(rn1)
3 print(rn1[0],rn1[1],rn1[2],rn1[3],rn1[4])
4 print()
5
6 rn2 = range(5,15)
7 print(rn2)
8 print(rn2[0],rn2[1],rn2[2],rn2[3],rn2[4])
9 print()
10
11 rn3 = range(5,50,5)
12 print(rn3)
13 print(rn3[0],rn3[1],rn3[2],rn3[3],rn3[4])
14
15
```

```
☞ range(0, 10)
0 1 2 3 4
```

```
range(5, 15)
5 6 7 8 9
```

```
range(5, 50, 5)
5 10 15 20 25
```

# What we'll cover in this lesson

---

1. What is a list?

**2. List functions**

3. Object types

4. Logical Operations

# Our sample list

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

Here we have a sample list (seawater) containing the 6 ions in seawater with the highest concentrations (Chloride, Sodium, Magnesium, Sulphate, Calcium, Potassium). Each item in the list has the ion name, the chemical symbol, and the percent of total ions (%). Below the sample list, lists containing the next 4 highest concentration ions are shown as well.

We will be using all of these lists to showcase common list functions.

# List functions

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

Adding items to a list:

# List functions

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

Adding items to a list:

**append()**

Adds a single item to the end of the list

```
1 seawater.append(bicarb)
2 print(seawater)
3
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14],
    ['Bicarbonate', 'HCO3', 0.41]]
```

# List functions

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

Adding items to a list:

**extend()**

Adds multiple items to the end of the list

```
1 seawater.extend([bicarb,bromide,borate,strontium])
2 print(seawater)
3
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14],
    ['Bicarbonate', 'HCO3', 0.41],
    ['Bromide', 'Br', 0.19],
    ['Borate', 'B(OH)4', 0.08],
    ['Strontium', 'Sr', 0.04]]
```

**Notice that the  
input has to be a  
list!**

# List functions

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

## append()

```
1 seawater.append([bicarb,bromide,borate,strontium])
2 print(seawater)
```

```
[['Chloride', 'Cl', 55.29],
 ['Sodium', 'Na', 30.74],
 ['Magnesium', 'Mg', 3.69],
 ['Sulphate', 'SO4', 7.75],
 ['Calcium', 'Ca', 1.18],
 ['Potassium', 'K', 1.14],
 [['Bicarbonate', 'HCO3', 0.41], ['Bromide', 'Br', 0.19], ['Borate', 'B(OH)4', 0.08], ['Strontium', 'Sr', 0.04]]]
```

## extend()

```
1 seawater.extend([bicarb,bromide,borate,strontium])
2 print(seawater)
3
```

```
[['Chloride', 'Cl', 55.29],
 ['Sodium', 'Na', 30.74],
 ['Magnesium', 'Mg', 3.69],
 ['Sulphate', 'SO4', 7.75],
 ['Calcium', 'Ca', 1.18],
 ['Potassium', 'K', 1.14],
 ['Bicarbonate', 'HCO3', 0.41],
 ['Bromide', 'Br', 0.19],
 ['Borate', 'B(OH)4', 0.08],
 ['Strontium', 'Sr', 0.04]]
```

**Double brackets here mean that the appended list of concentrations was added as a single item. This is not what we want!**



# List functions

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

Adding items to a list:

+

Concatenates the items from two lists

```
1 seawater_new = seawater + [bicarb, bromide, borate, strontium]
2 print(seawater_new)
3
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14],
    ['Bicarbonate', 'HCO3', 0.41],
    ['Bromide', 'Br', 0.19],
    ['Borate', 'B(OH)4', 0.08],
    ['Strontium', 'Sr', 0.04]]
```

# List functions

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

Adding items to a list:

+

Concatenates the items from two lists

```
1 seawater_new = seawater + bicarb
2 print(seawater_new)
3 print()
4
5 seawater_new = seawater + [bicarb]
6 print(seawater_new)
```

```
↳ [['Chloride', 'Cl', 55.29], ['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74], ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69], ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75], ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18], ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14], ['Potassium', 'K', 1.14],
    Bicarbonate,
    HCO3,
    0.41]
```

# List functions

Adding items to a list:

**insert()**

Adds a single item to a given index in the list

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

```
1 seawater.extend([bicarb,bromide,strontium])
2 print(seawater)
3 print()
4
5 seawater.insert(8,borate)
6 print(seawater)
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14],
    ['Bicarbonate', 'HCO3', 0.41],
    ['Bromide', 'Br', 0.19],
    ['Strontium', 'Sr', 0.04]]
    [['Chloride', 'Cl', 55.29],
     ['Sodium', 'Na', 30.74],
     ['Magnesium', 'Mg', 3.69],
     ['Sulphate', 'SO4', 7.75],
     ['Calcium', 'Ca', 1.18],
     ['Potassium', 'K', 1.14],
     ['Bicarbonate', 'HCO3', 0.41],
     ['Bromide', 'Br', 0.19],
     ['Borate', 'B(OH)4', 0.08],
     ['Strontium', 'Sr', 0.04]]
```

# List functions

## Removing items from a list:

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

# List functions

Removing items from a list:

**remove()**

Deletes the first occurrence of a given item

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
```

```
10
11 print(seawater)
```

```
13 # The next highest concentrations
```

```
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
```

```
15 bromide = ['Bromide', 'Br', 0.19]
```

```
16 borate = ['Borate', 'B(OH)4', 0.08]
```

```
17 strontium = ['Strontium', 'Sr', 0.04]
```

```
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

```
1 seawater.insert(0,bicarb)
2 seawater.append(bicarb)
3 print(seawater)
4 print()
5
6 seawater.remove(bicarb)
7 print(seawater)
8
```

**Notice that the input must be in the list or this does not work**

```
↳ [['Bicarbonate', 'HCO3', 0.41],
    ['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14],
    ['Bicarbonate', 'HCO3', 0.41]]
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14],
    ['Bicarbonate', 'HCO3', 0.41]]
```

# List functions

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

## Removing items from a list:

### **del**

Deletes the items in a given index.

Can also delete the whole list (and any other objects)!

```
1 del seawater[3:]
2 print(seawater)
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69]]
```

```
1 del seawater
2 print(seawater)
```

```
↳ -----
NameError                                Traceback (most recent call last)
<ipython-input-156-210b9e2119a7> in <module>()
      1 del seawater
----> 2 print(seawater)

NameError: name 'seawater' is not defined
```

# List functions

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

Removing items from a list:

**pop()**

Removes and outputs an indexed item (default= -1)

```
1 second_highest = seawater.pop(1)
2 print(seawater)
3 print()
4
5 first_highest = seawater.pop(0)
6 print(seawater)
7 print()
8
9 print('1st:', first_highest)
10 print('2nd:', second_highest)
```

```
[[ 'Chloride', 'Cl', 55.29],
 [ 'Magnesium', 'Mg', 3.69],
 [ 'Sulphate', 'SO4', 7.75],
 [ 'Calcium', 'Ca', 1.18],
 [ 'Potassium', 'K', 1.14]]
```

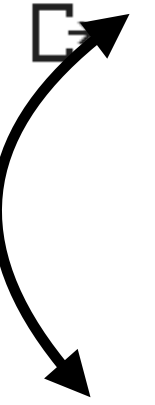
```
[[ 'Magnesium', 'Mg', 3.69],
 [ 'Sulphate', 'SO4', 7.75],
 [ 'Calcium', 'Ca', 1.18],
 [ 'Potassium', 'K', 1.14]]
```

```
1st: ['Chloride', 'Cl', 55.29]
2nd: ['Sodium', 'Na', 30.74]
```

# List functions

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```



Reversing a list:

**reverse()**

Reverses the order that a list is in

```
1 seawater.reverse()
2 print(seawater)
```

```
↳ [['Potassium', 'K', 1.14],
    ['Calcium', 'Ca', 1.18],
    ['Sulphate', 'SO4', 7.75],
    ['Magnesium', 'Mg', 3.69],
    ['Sodium', 'Na', 30.74],
    ['Chloride', 'Cl', 55.29]]
```



# List functions

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

Setting a variable equal to a list and then changing the list, changes the variable too.

```
1 del seawater[3:]
2 top3 = seawater
3
4 print(seawater)
5 print()
6 print(top3)
7 print()
8
9 seawater.extend([bicarb, bromide, borate])
10
11 print(seawater)
12 print()
13 print(top3)
14 print()
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69]]

[['Chloride', 'Cl', 55.29],
 ['Sodium', 'Na', 30.74],
 ['Magnesium', 'Mg', 3.69]]

[['Chloride', 'Cl', 55.29],
 ['Sodium', 'Na', 30.74],
 ['Magnesium', 'Mg', 3.69],
 ['Bicarbonate', 'HCO3', 0.41],
 ['Bromide', 'Br', 0.19],
 ['Borate', 'B(OH)4', 0.08]]

[['Chloride', 'Cl', 55.29],
 ['Sodium', 'Na', 30.74],
 ['Magnesium', 'Mg', 3.69],
 ['Bicarbonate', 'HCO3', 0.41],
 ['Bromide', 'Br', 0.19],
 ['Borate', 'B(OH)4', 0.08]]
```

# List functions

```
1 # Seawater composition:
2 # name, symbol, % of total ions (35 PSU)
3
4 seawater = [['Chloride', 'Cl', 55.29],
5 ['Sodium', 'Na', 30.74],
6 ['Magnesium', 'Mg', 3.69],
7 ['Sulphate', 'SO4', 7.75],
8 ['Calcium', 'Ca', 1.18],
9 ['Potassium', 'K', 1.14]]
10
11 print(seawater)
12
13 # The next highest concentrations
14 bicarb = ['Bicarbonate', 'HCO3', 0.41]
15 bromide = ['Bromide', 'Br', 0.19]
16 borate = ['Borate', 'B(OH)4', 0.08]
17 strontium = ['Strontium', 'Sr', 0.04]
18
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Sulphate', 'SO4', 7.75],
    ['Calcium', 'Ca', 1.18],
    ['Potassium', 'K', 1.14]]
```

Setting a variable equal to a list and then changing the list, changes the variable too. Avoid this by using **copy()**

```
1 del seawater[3:]
2 top3 = seawater.copy()
3
4 print(seawater)
5 print()
6 print(top3)
7 print()
8
9 seawater.extend([bicarb, bromide, borate])
10
11 print(seawater)
12 print()
13 print(top3)
14 print()
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69]]
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69]]
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69],
    ['Bicarbonate', 'HCO3', 0.41],
    ['Bromide', 'Br', 0.19],
    ['Borate', 'B(OH)4', 0.08]]
```

```
↳ [['Chloride', 'Cl', 55.29],
    ['Sodium', 'Na', 30.74],
    ['Magnesium', 'Mg', 3.69]]
```

# List functions

---

When a list has only strings in it, you can combine the different string items into a single string object.

**join()**

**split()**

```
1 seawater_top3 = ['Chloride', 'Sodium', 'Magnesium']
2
3 delimiter = ' '
4
5 seawater_string = delimiter.join(seawater_top3)
6 print(seawater_string)
7
8
9 seawater_top3 = seawater_string.split(delimiter)
10 print(seawater_top3)
```

```
☞ Chloride Sodium Magnesium
   ['Chloride', 'Sodium', 'Magnesium']
```

# List functions

---

When a list has only strings in it, you can combine the different string items into a single string object.

**join()**

**split()**

```
1 seawater_top3 = ['Chloride', 'Sodium', 'Magnesium']
2
3 delimiter = '?'
4
5 seawater_string = delimiter.join(seawater_top3)
6 print(seawater_string)
7
8
9 seawater_top3 = seawater_string.split(delimiter)
10 print(seawater_top3)
```

```
☞ Chloride?Sodium?Magnesium
   ['Chloride', 'Sodium', 'Magnesium']
```

# List functions

append	Put single item on the end	-	<code>list.append( object )</code>
extend	Put multiple items on the end	Items must be put into a list	<code>list.extend( [ object ] )</code>
+	Concatenate 2 lists	Be cautious of concatenating lists within lists	<code>new_list = list1 + list2</code>
insert	Put an item in at a specific index	Specify the index value first	<code>list.insert( index, object )</code>
remove	Delete the first occurrence of an item	Object must be in the list	<code>list.remove( object )</code>
del	Delete a slice (using indexing)	Can remove whole objects too!	<code>del list</code>
pop	Remove and output and item at a specific index (default: -1)	Keep track of how your index values change	<code>list.pop(index)</code>
reverse	Reverse the order of the list	-	<code>list.reverse( )</code>
copy	Create a separate copy of a list	This helps to keep a version of the list that is “original”	<code>new_list = list.copy( )</code>
join	Join the strings in a list into a single string	Can only be used if the list has just strings in it	<code>' '.join(list)</code>

# What we'll cover in this lesson

---

1. What is a list?
2. List functions
- 3. Object types**
4. Logical Operations

# Finding the object type

## Object types so far:

Integer }  
Float } Numbers  
Complex }  
  
Boolean  
  
String  
  
List

```
1 # Create example variables
2 int_ex = 492
3
4 float_ex = 3.14159
5
6 comp_ex = 1.5 + 2.4j
7
8 bool_ex = True
9
10 string_ex = 'carrots'
11
12 list_ex = [3, 2, 1, 'Hi', True]
13
14 # Print statements
15
16 print(type(int_ex)) <class 'int'>
17 print(type(float_ex)) <class 'float'>
18 print(type(comp_ex)) <class 'complex'>
19 print(type(bool_ex)) <class 'bool'>
20 print(type(string_ex)) <class 'str'>
21 print(type(list_ex)) <class 'list'>
22
```

To find out what type of object a variable is, use the **type()** function.

# Changing object type

**Object types so far:**

To change an object into an integer, use the function: **int()**

Integer

Float

Complex

Boolean

String

List

```
# Create some variables
float1 = 12.6
string2 = '15'
bool3 = False

# Change them to integers
int1 = int(float1)
int2 = int(string2)
int3 = int(bool3)

print(int1, int2, int3)
print(type(int1), type(int2), type(int3))
```

← Notice that changing a float to an integer drops the decimal

**Cases that this does not work:**

- If your variable is a complex object
- If your variable is a list
- If your variable is a non-numeric string
- If your variable is a numeric string with a decimal

```
☞ 12 15 0
   <class 'int'> <class 'int'> <class 'int'>
```



# Changing object type

---

**Object types so far:**

To change an object into a float, use the function: **float()**

Integer

**Float**

Complex

Boolean

String

List

```
# Create some variables
int1 = 575
string2 = '16.92'
bool3 = False

# Change them to floats
float1 = float(int1)
float2 = float(string2)
float3 = float(bool3)

print(float1, float2, float3)
print(type(float1), type(float2), type(float3))
```

```
↳ 575.0 16.92 0.0
   <class 'float'> <class 'float'> <class 'float'>
```

**Cases that this does not work:**

- If your variable is a complex object
- If your variable is a list
- If your variable is a non-numeric string

# Changing object type

---

**Object types so far:**

To change an object into a complex, use the function: **complex()**

Integer

Float

**Complex**

Boolean

String

List

```
# Create some variables
int1 = 575
string2 = '16.92'
bool3 = False

# Change them to complex
complex1 = complex(int1)
complex2 = complex(string2)
complex3 = complex(bool3)

print(complex1, complex2, complex3)
print(type(complex1), type(complex2), type(complex3))
```

```
↳ (575+0j) (16.92+0j) 0j
   <class 'complex'> <class 'complex'> <class 'complex'>
```

**Cases that this does not work:**

- If your variable is a list
- If your variable is a non-numeric string

# Changing object type

**Object types so far:**

To change an object into a boolean, use the function: **bool()**

Integer

Float

Complex

**Boolean**

String

List

```
1 # Create some variables
2 int1 = 575
3 string2 = '16.92'
4 list3 = []
5 float4 = 0.0
6
7 # Change them to booleans
8 bool1 = bool(int1)
9 bool2 = bool(string2)
10 bool3 = bool(list3)
11 bool4 = bool(float4)
12
13 print(bool1, bool2, bool3, bool4)
14 print(type(bool1), type(bool2), type(bool3), type(bool4))
```

```
☞ True True False False ←
<class 'bool'> <class 'bool'> <class 'bool'> <class 'bool'>
```

**Cases that this does not work:**

- 

**Only objects that are empty  
or zero will produce a False.  
All other objects are True.**

# Changing object type

---

## Object types so far:

To change an object into a string, use the function: **str()**

Integer

Float

Complex

Boolean

String

List

```
# Create some variables
int1 = 575
complex2 = 6.8+0.2j
bool3 = False
float4 = 0.0

# Change them to strings
str1 = str(int1)
str2 = str(complex2)
str3 = str(bool3)
str4 = str(float4)

print(str1, str2, str3, str4)
print(type(str1), type(str2), type(str3), type(str4))
```

```
☞ 575 (6.8+0.2j) False 0.0
   <class 'str'> <class 'str'> <class 'str'> <class 'str'>
```

## Cases that this does not work:

-

# Changing object type

## Object types so far:

Integer

Float

Complex

Boolean

String

List

To change any object into a list, use square brackets [ ]

To change iterable objects into a string, use **list()**

Basic definition: an object with a length (e.g. strings, lists)

```
[35] 1 # Create some variables
      2 int1 = 575
      3 string2 = '16.92'
      4 bool3 = True
      5 float4 = 0.0
      6
      7 # Change them to booleans
      8 list1 = [int1]
      9 list2 = list(string2)
     10 list2_5 = [string2]
     11 list3 = [bool3]
     12 list4 = [float4]
     13
     14 print(list1, list2, list2_5, list3, list4)
     15 print(type(list1), type(list2), type(list2_5), type(list3), type(list4))
```

```
↳ [575] ['1', '6', '.', '9', '2'] ['16.92'] [True] [0.0]
   <class 'list'> <class 'list'> <class 'list'> <class 'list'> <class 'list'>
```

## Cases that this does not work:

-

# What we'll cover in this lesson

---

1. What is a list?
2. List functions
3. Object types
- 4. Logical Operations**

# Logical operations

---

## Comparison operators

Operation	
==	Equal
!=	Not Equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Using the comparison operators (lesson #2) we can add more parameters to our comparisons using logical operators...

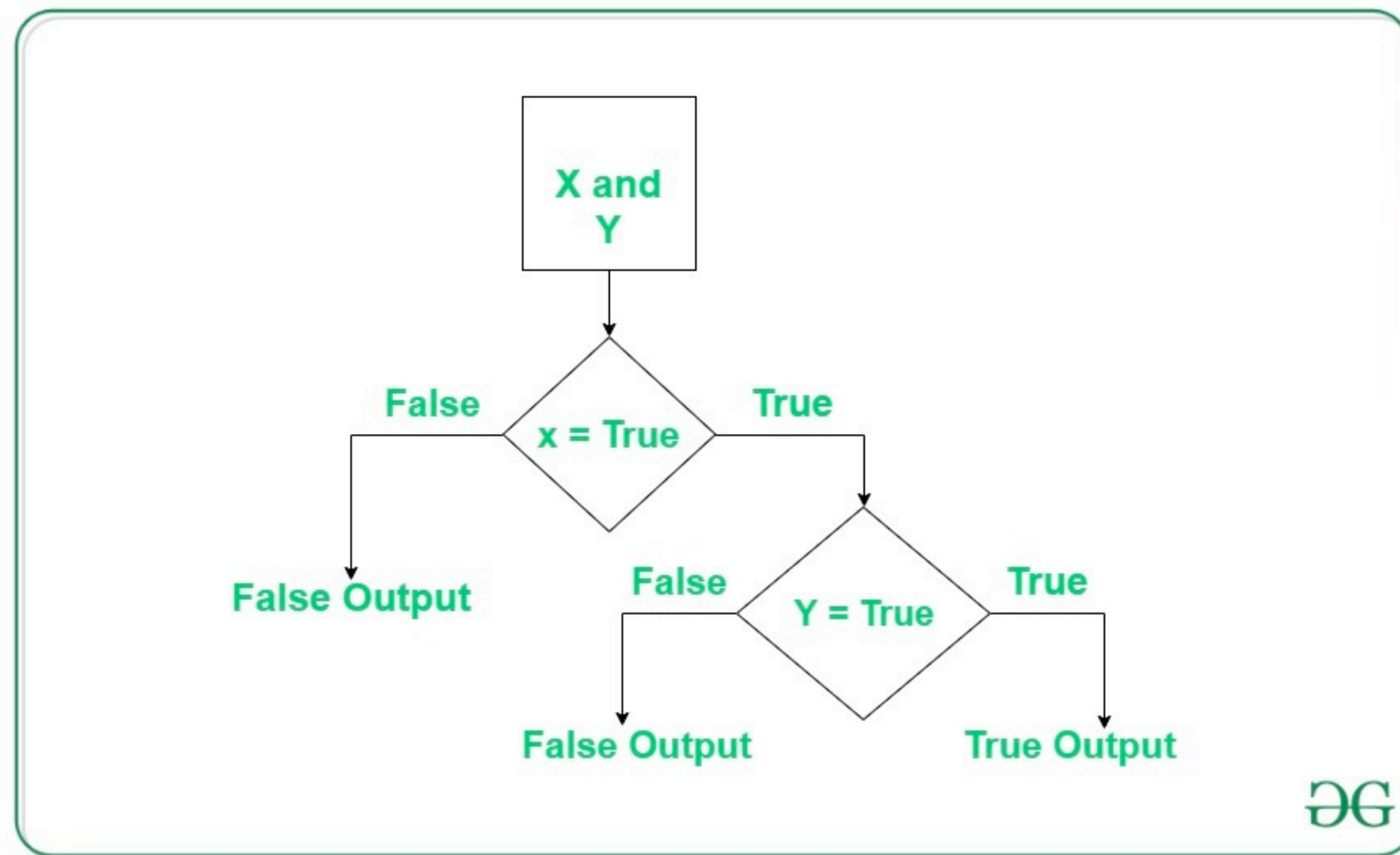
# Logical operations

Comparison operators

Operation	
==	Equal
!=	Not Equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Using the comparison operators (lesson #2) we can add more parameters to our comparisons using logical operators...

**and**





# Logical operations

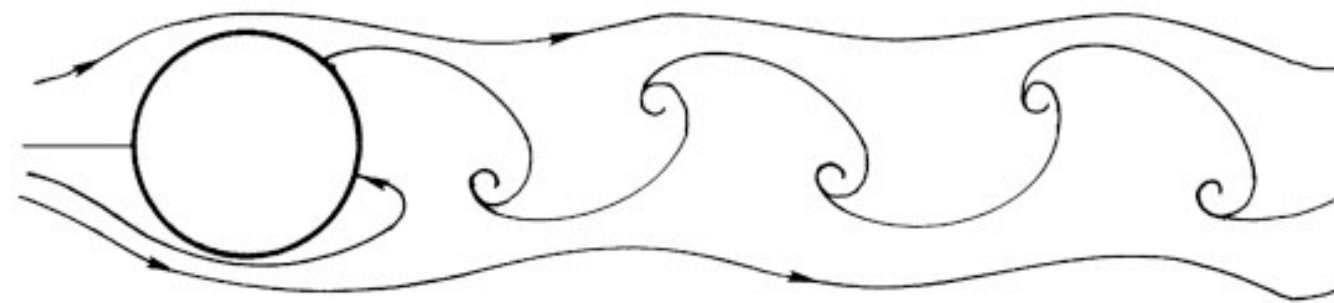
## Comparison operators

Operation	
$==$	Equal
$!=$	Not Equal
$>$	Greater than
$>=$	Greater than or equal to
$<$	Less than
$<=$	Less than or equal to

Using the comparison operators (lesson #2) we can add more parameters to our comparisons using logical operators...

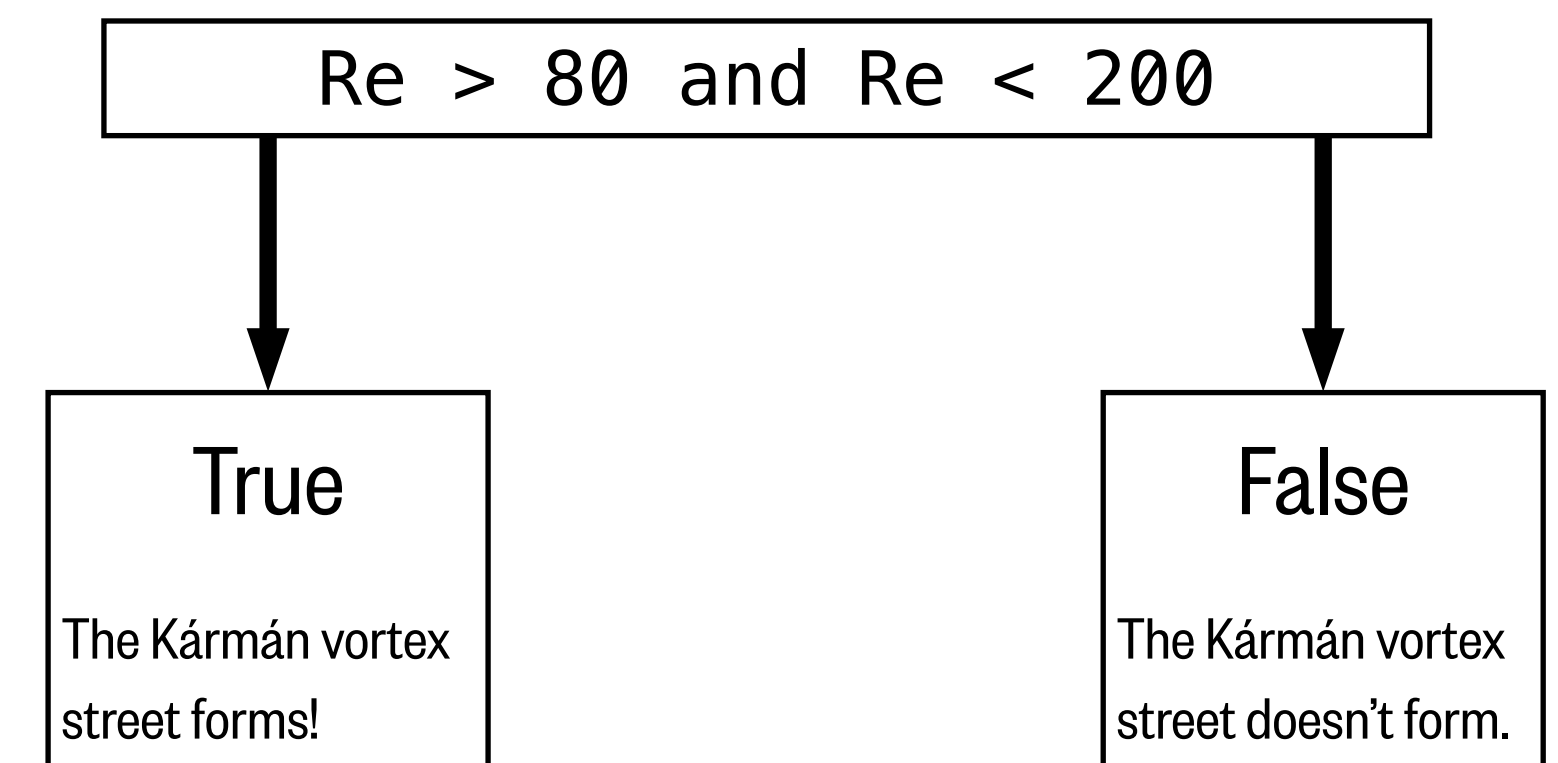
## and

Example: Reynold's number



The relationship between the Reynold's number and the turbulence of a flow have been well established. The Kármán vortex street is estimated to occur when the Reynold's number is between 80 - 200.

Given an unknown Reynold's number, we can test if the Kármán vortex street will occur.



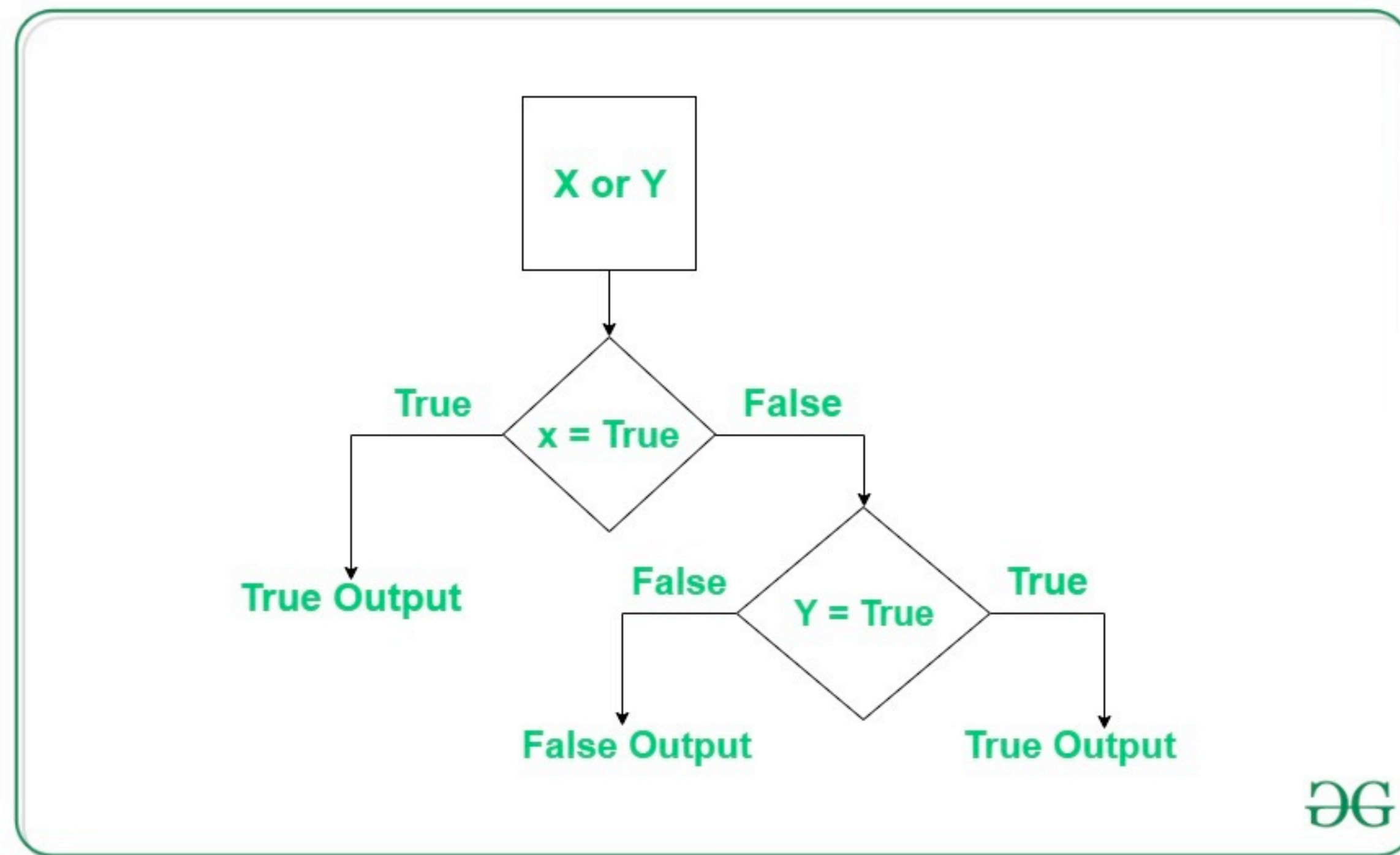
# Logical operations

## Comparison operators

Operation	
==	Equal
!=	Not Equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Using the comparison operators (lesson #2) we can add more parameters to our comparisons using logical operators...

**or**



# Logical operations

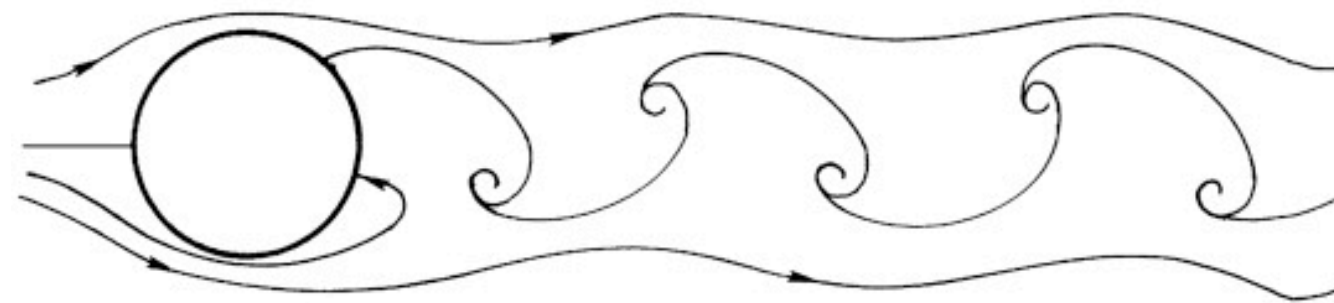
## Comparison operators

Operation	
==	Equal
!=	Not Equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Using the comparison operators (lesson #2) we can add more parameters to our comparisons using logical operators...

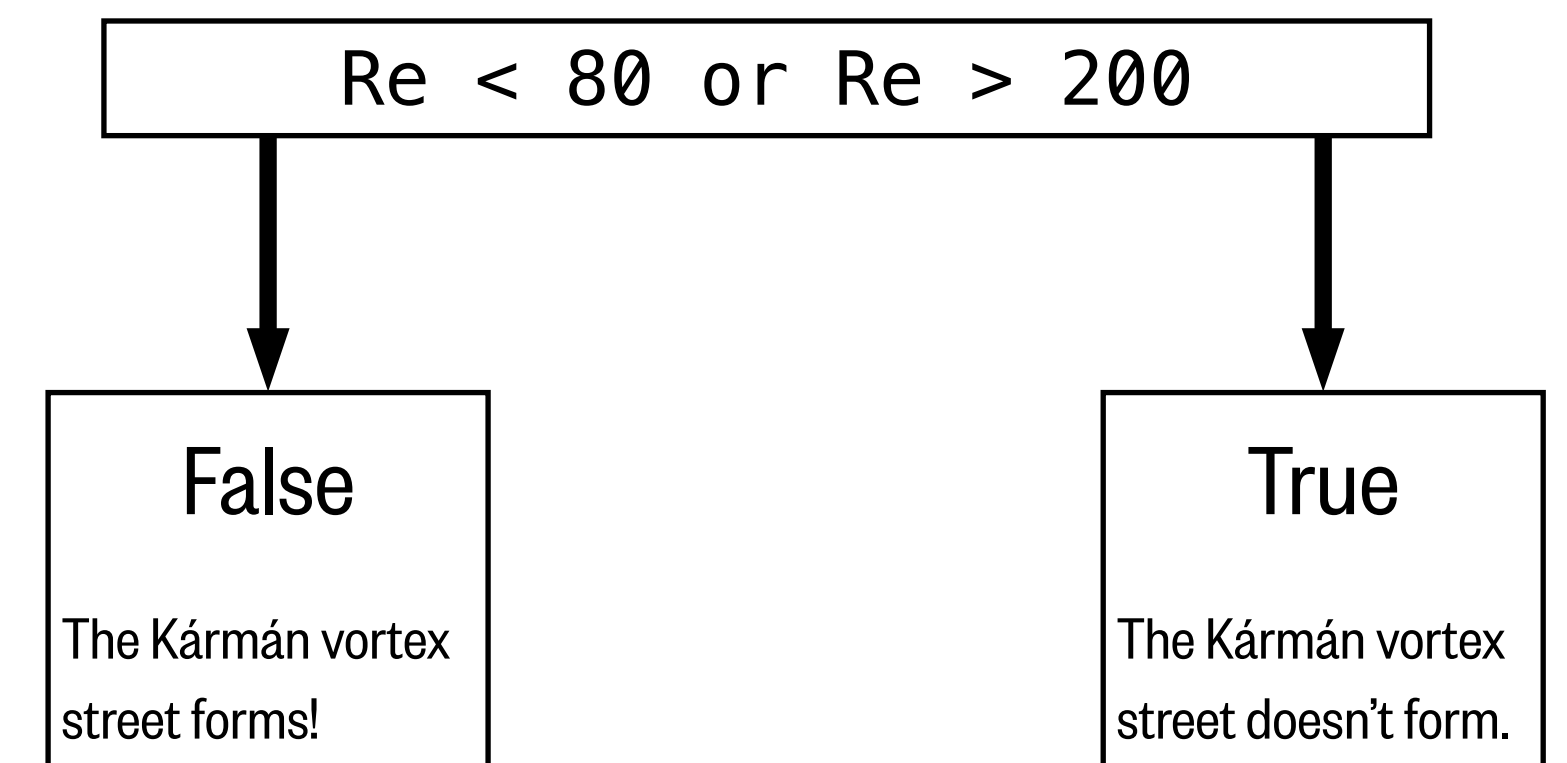
**or**

Example: Reynold's number



The relationship between the Reynold's number and the turbulence of a flow have been well established. The Kármán vortex street is estimated to occur when the Reynold's number is between 80 - 200.

Given an unknown Reynold's number, we can test if the Kármán vortex street will occur.



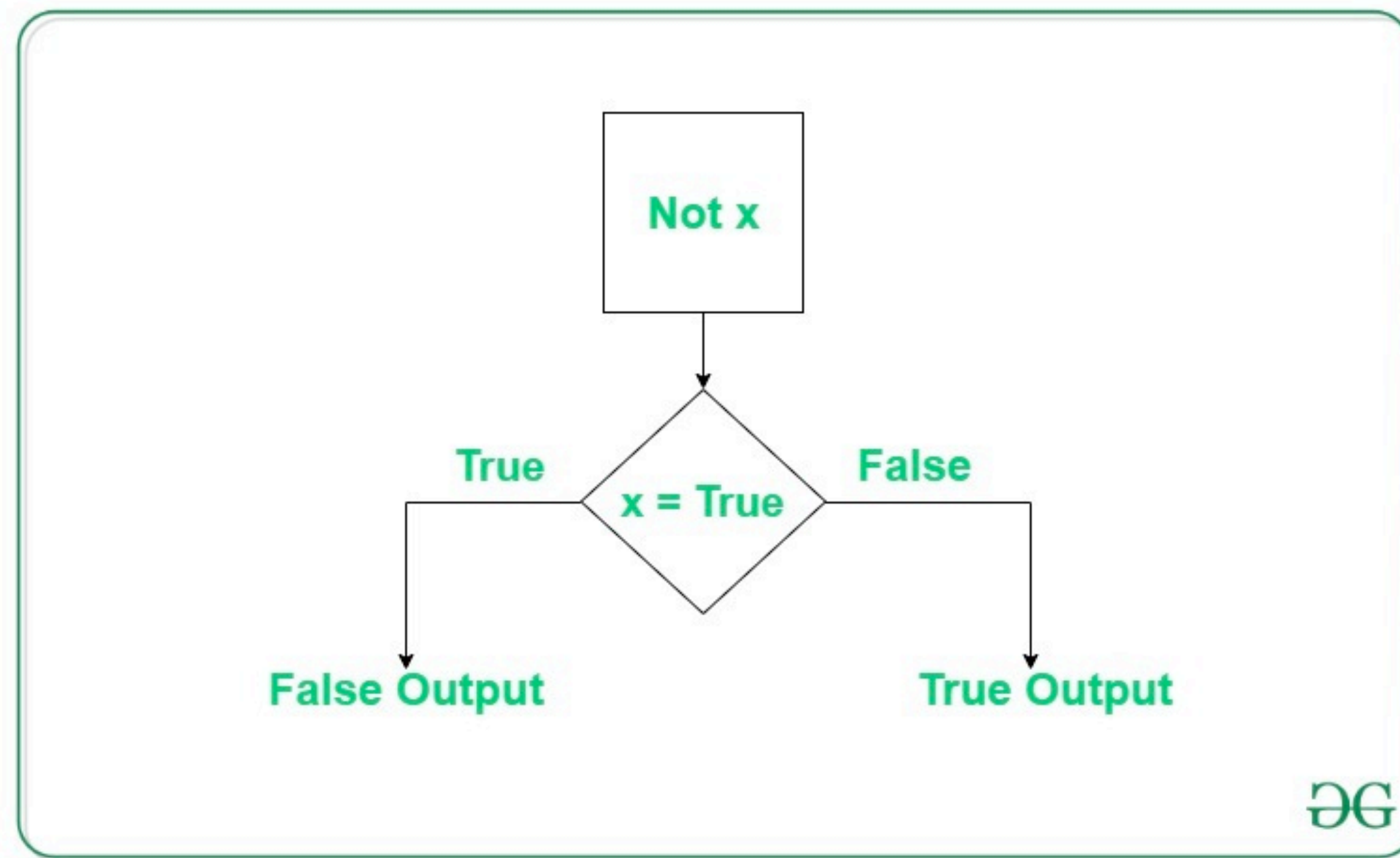
# Logical operations

## Comparison operators

Operation	
==	Equal
!=	Not Equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Using the comparison operators (lesson #2) we can add more parameters to our comparisons using logical operators...

## not



not True	not False
False	True

# is

Test if two variables refer to the same object. This is useful when you assign variables to lists.

If you assign a variable to a list, it “is” that list  
(no matter what)

```
1 sample_list = ['Apples', 'Oranges', 'Bananas']
2 list_var = sample_list
3
4 print(list_var is sample_list)
5
6 sample_list.append('Strawberries')
7 print(list_var is sample_list)
8
```

```
☞ True
True
```

If you assign a variable to a copy of a list, it “is”  
not that list

```
1 sample_list = ['Apples', 'Oranges', 'Bananas']
2 print(sample_list)
3 print()
4
5 list_var = sample_list.copy()
6
7 print(list_var is sample_list)
8
9 sample_list.append('Strawberries')
10 print(list_var is sample_list)
11
12 print()
13 print(sample_list)
14
```

```
☞ ['Apples', 'Oranges', 'Bananas']
```

```
False
False
```

```
['Apples', 'Oranges', 'Bananas', 'Strawberries']
```

If you assign a variable to an identical, but  
separate list, it “is” not that list

```
1 sample_list1 = ['Apples', 'Oranges', 'Bananas']
2 print(sample_list1)
3 print()
4
5 sample_list2 = ['Apples', 'Oranges', 'Bananas']
6 print(sample_list2)
7 print()
8
9 print(sample_list1 is sample_list2)
10
```

```
☞ ['Apples', 'Oranges', 'Bananas']
```

```
['Apples', 'Oranges', 'Bananas']
```

```
False
```

# Resources

---

Seawater ions - <http://www.marinebio.net/marinescience/02ocean/swcomposition.htm>

Logical operator flowcharts - <https://www.geeksforgeeks.org/python-logical-operators-with-examples-improvement-needed/>

Kármán vortex street - <https://www.sciencedirect.com/topics/engineering/creeping-flow>