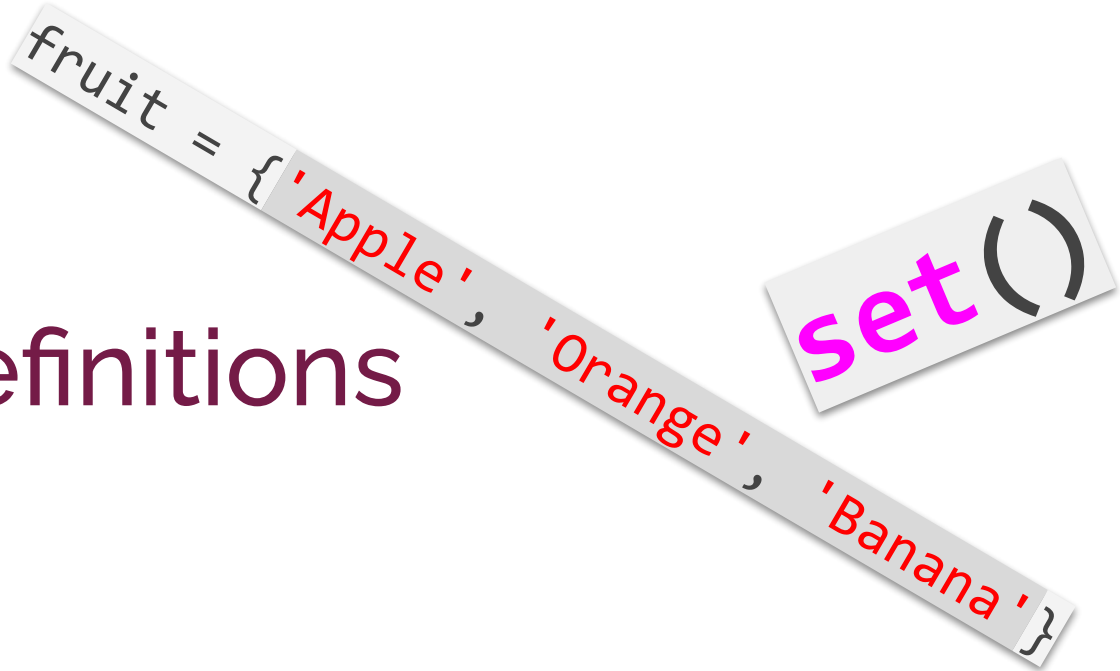
Sets

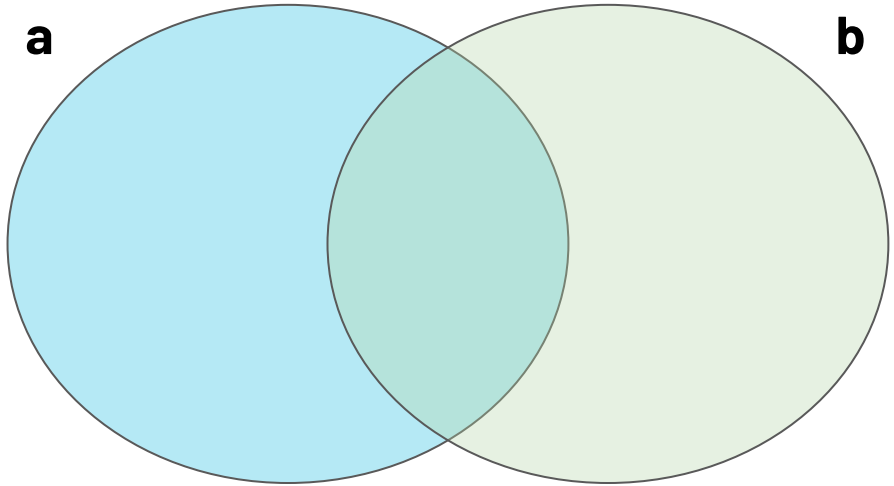
Table of Contents

Definitions



Definitions

- No repetition
- Math operations
  - union
  - intersection
  - difference
- Unordered elements



Creating a `set`

Creating a `set`

- We have two basic ways to create a set.

- `{}`
- `set()`

```
set_1 = {'red', 'blue', 'pink', 'red'}
colors = 'red', 'blue', 'pink', 'red'
set_2 = set(colors)
print(set_1)
```

```
{'blue', 'pink', 'red'}
```

Creating a `set`

- A `set` can be created by enclosing values, separated by commas, in curly braces 👉 `{}`.
- Another way to create a `set` is to call the `set()` function.

- `{}`
- `set()`

```
set_1 = {'red', 'blue', 'pink', 'red'}
colors = 'red', 'blue', 'pink', 'red'
set_2 = set(colors)
print(set_1)
print(set_2)
```

```
{'red', 'blue', 'pink'}
{'red', 'blue', 'pink'}
```

⚠️ different order from the previous slide

Creating a `set` (review of pre-class)

- Here is an example of creating an empty `set`:

input :

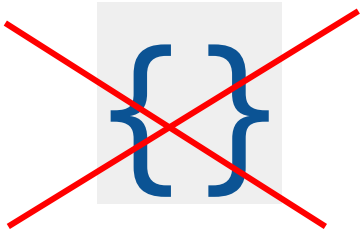
```
1  empty_set = set()
2
3  print(type(empty_set))
4
```

output :

```
1  <class 'set'>
2
```

Creating a `set`

- Creating an empty set



To create an empty `set`, you **can not** use 👉 `{}`. The only way to create an empty set is `set()` function.

Creating a `set` (review of pre-class)

```
1  flower_list = ['rose', 'violet', 'carnation', 'rose', 'orchid', 'rose', 'orchid']
2  flowerset = set(flower_list)
3  flowerlist = list(flowerset)
4
5  print(flowerset)
6  print(flowerlist)
7
```

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

Students, write your response!

Creating a `set` (review of pre-class)

```
1  flower_list = ['rose', 'violet', 'carnation', 'rose', 'orchid', 'rose', 'orchid']
2  flowerset = set(flower_list)
3  flowerlist = list(flowerset)
4
5  print(flowerset)
6  print(flowerlist)
7
```

```
1  {'orchid', 'carnation', 'violet', 'rose'}
2  ['orchid', 'carnation', 'violet', 'rose']
3
```

Creating a `set` (review of pre-class)

- **Task** :
  - Do these two sets give the same output and why?

```
a = {'carnation', 'orchid', 'rose', 'violet'}
```

👇👆

```
b = {'rose', 'orchid', 'rose', 'violet', 'carnation'}
```

Students choose an option

Creating a `set`

- The Answer is : **`True`**

```
{'carnation', 'orchid', 'rose', 'violet'}

{'rose', 'orchid', 'rose', 'violet', 'carnation'}
```

Main Operations with Sets

Main Operations with `sets` (review)

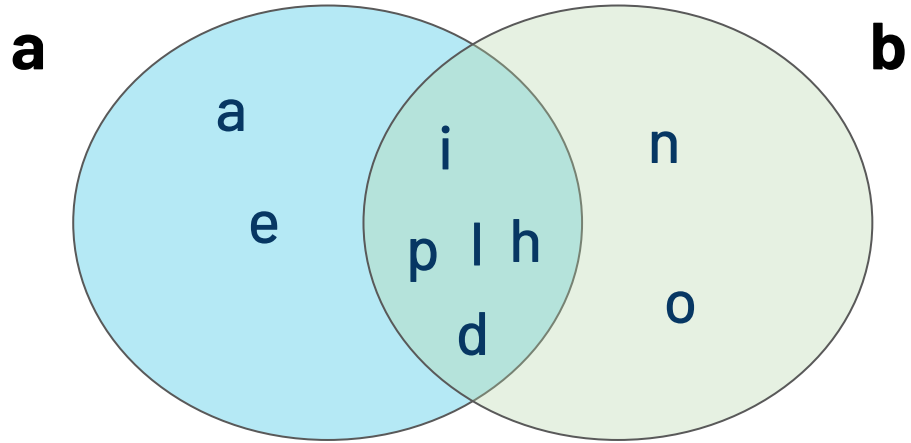
- The methods that can be used with `sets` :

- `.add()` : Adds a new item to the set.
- `.remove()` : Allows us to delete an item.
- `.intersection()` : Returns the intersection of two sets.
- `.union()` : Returns the unification of two sets.
- `.difference()` : Gets the difference of two sets.

Main Operations with `sets`

- Let's take a look these two sets below :

```
a = set('philadelphia')
b = set('dolphin')
```

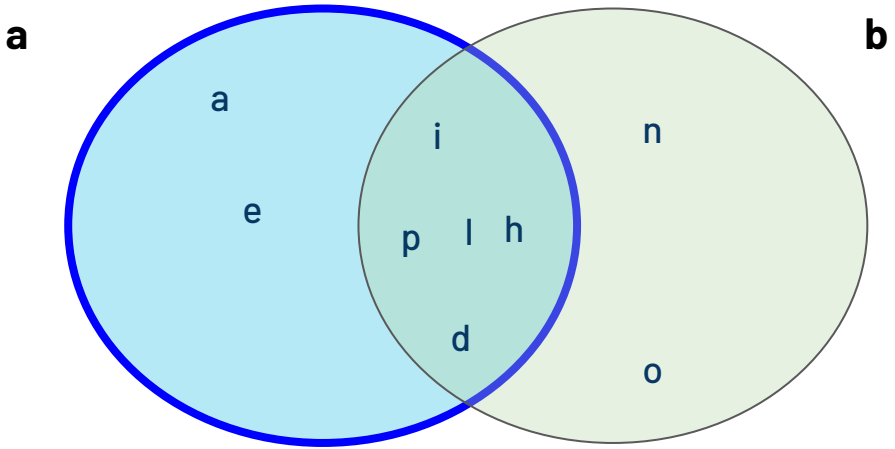


Main Operations with `sets`

- Let's take a look these two sets below :

```
a = set('philadelphia')
print(a)
```

```
{'a', 'e', 'i', 'd', 'l', 'p', 'h'}
```

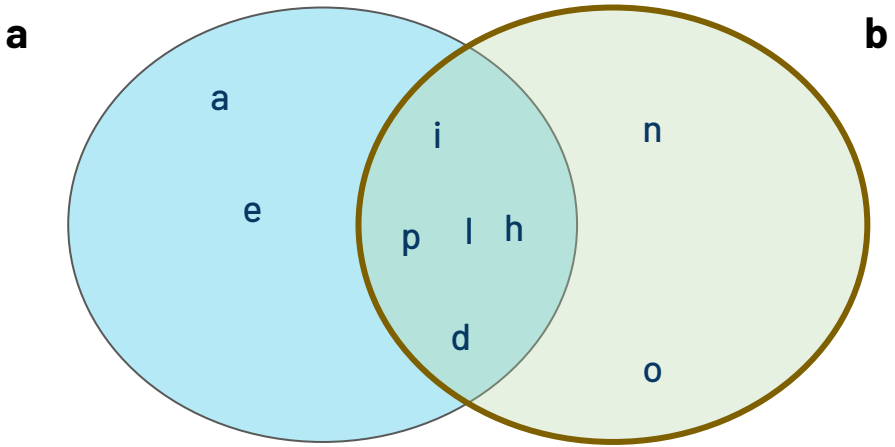


Main Operations with `sets`

- Let's take a look these two sets below :

```
b = set('dolphin')
print(b)
```

```
{'d', 'l', 'o', 'p', 'n', 'i', 'h'}
```



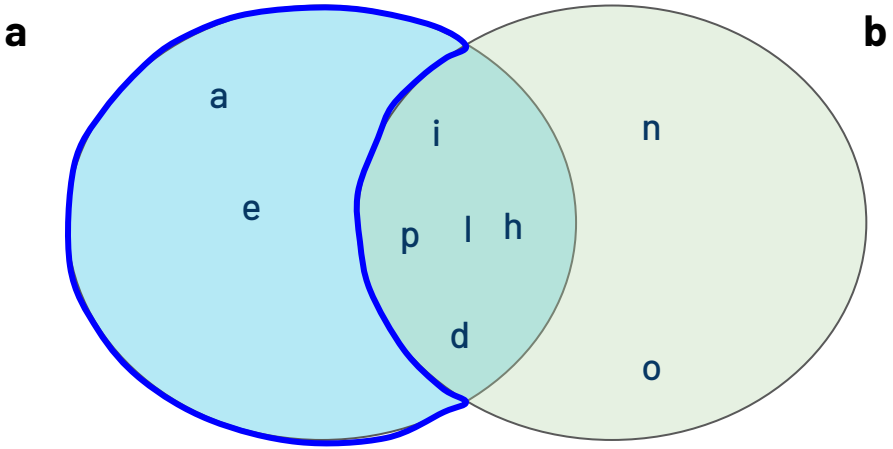
Main Operations with `sets`

- Basic `set` operations :

`.difference(arg)`

```
print(a - b)
print(a.difference(b))
```

```
{'a', 'e'}
```



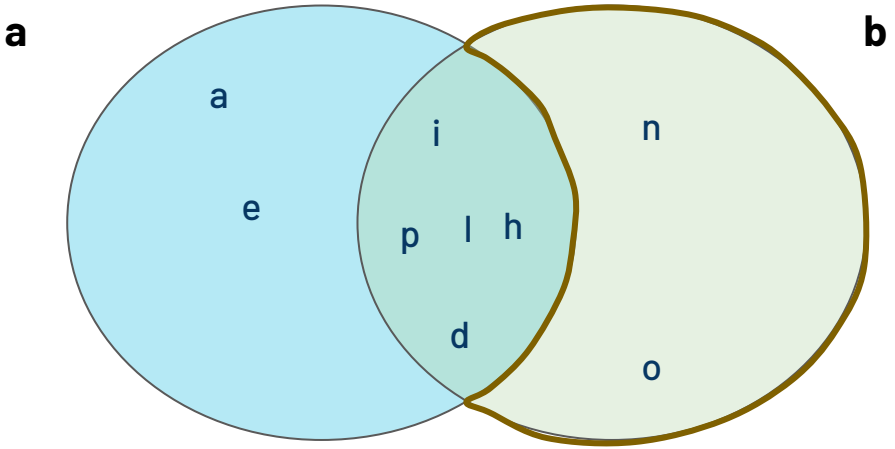
Main Operations with `sets`

- Basic `set` operations :

`.difference(arg)`

```
print(b - a)
print(b.difference(a))
```

```
{'n', 'o'}
```



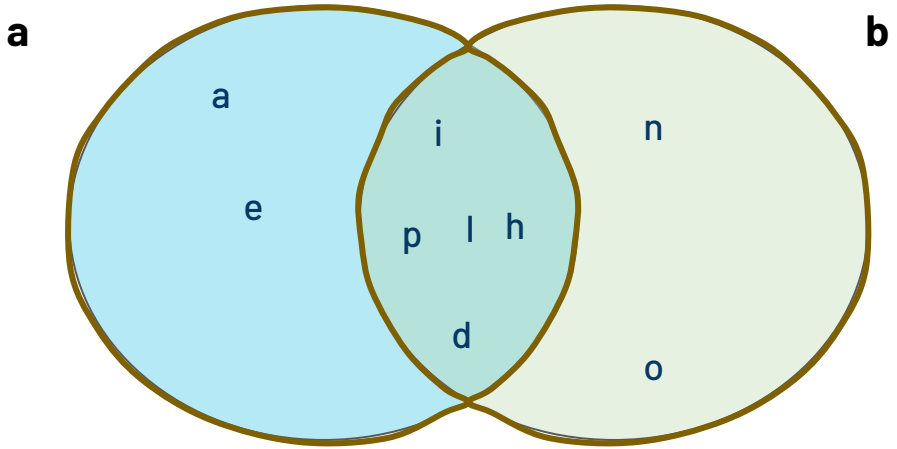
Main Operations with `sets`

- Basic `set` operations :

`.union(arg)`

```
print(a | b)
print(a.union(b))
```

```
{'p', 'h', 'i', 'l', 'd', 'o', 'n', 'a', 'e'}
```



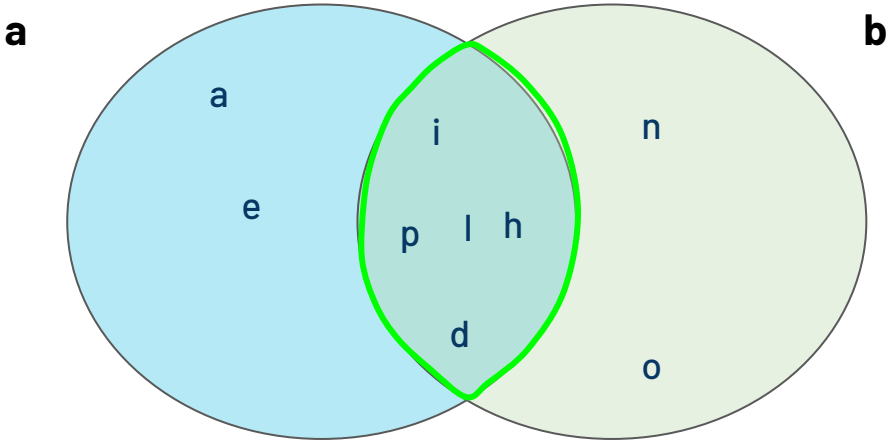
Main Operations with `sets`

- Basic `set` operations :

`.intersection(arg)`

```
print(a & b)
print(a.intersection(b))
```

```
{'p', 'h', 'i', 'l', 'd'}
```



Creating a `set`

- Task :
  - Let's create a `set` from which `str` type of the current date?
  - Date style would be "`mm/dd/yyyy`".
  - Creating a **set**, use both `set()` function and `{}` then figure out the results.

Students, write your response!

Creating a `set`

- The solution:

```
a = set('09/01/2023')
b = {'09/01/2023'}
print(a)
print(b)
```

```
{'9', '3', '0', '2', '1', '/'}
{'09/01/2023'}
```

Creating a `set`

- **Task** :
  Given a `list`, create a `set` to select and print the **unique** elements of the it.

```
given_list = [1, 2, 3, 3, 3, 3, 4, 4, 5, 5]
```

Creating a `set`

- **The code might be like** :

```
given_list = [1, 2, 3, 3, 3, 3, 4, 4, 5, 5]

unique = set(given_list)

print(unique)
```

```
{1, 2, 3, 4, 5}
```

Discuss in-class! Could you do the same thing using only curly braces `{}` instead of `set()` function?

Creating a `set`

- **Task** :
  -Create two `set`s of string data from the capitals of the **USA** and **New Zealand**. (**e.g**: `'Madrid'` → `convert into a set`)
  -Perform all set operations.
    - Intersection
    - Union
    - Difference

Creating a `set`

- **The code might be like** :

```
usa_capt = set('Washington')
nz_capt = set('Wellington')

print(usa_capt)
print(nz_capt)
```

```
{'h', 'W', 'a', 'o', 's', 'n', 'g', 'i', 't'}
{'W', 'o', 'l', 'e', 'n', 'g', 'i', 't'}
```

Creating a `set`

- **The code might be like** :

```
usa_capt = set('Washington')
nz_capt = set('Wellington')

print(usa_capt - nz_capt)
print(usa_capt.difference(nz_capt))
```

```
{'s', 'h', 'a'}
{'s', 'h', 'a'}
```

Creating a `set`

- **The code might be like** :

```
usa_capt = set('Washington')
nz_capt = set('Wellington')

print(nz_capt - usa_capt)
print(nz_capt.difference(usa_capt))
```

```
{'l', 'e'}
{'l', 'e'}
```

Creating a `set`

- **The code might be like** :

```
usa_capt = set('Washington')
nz_capt = set('Wellington')

print(nz_capt & usa_capt)
print(nz_capt.intersection(usa_capt))
```

```
{'i', 'o', 'g', 'n', 't', 'W'}
{'i', 'o', 'g', 'n', 't', 'W'}
```

frozenset()

- **Frozen set is just an immutable version of a Python set:**

```
usa_capt = set('Washington')
dondur_capt = frozenset(usa_capt) #elements cannot be changed
print(dondur_capt)
dondur_capt.add('z')   ## ?
print(dondur_capt)
```

```
frozenset({'g', 's', 'i', 'o', 't', 'n', 'a', 'h', 'W'})
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-68-9e416fb52e55> in <module>()
      2 dondur_capt = frozenset(usa_capt) #elemanları değiştirlemeyen bir küme
      3 print(dondur_capt)
----> 4 dondur_capt.add('z')
      5 print(dondur_capt)

AttributeError: 'frozenset' object has no attribute 'add'
```

List

mutable
ordered
unhashable
iterable
duplicate

Tuple

immutable
ordered
hashable
iterable
duplicate

Dict

mutable
unordered
unhashable
no duplicate

Set

mutable
unordered
unhashable
iterable
no duplicate

Mutable or Immutable ?

List of immutable types:

```
int, float, decimal, complex, bool, string, tuple, range, frozenset, bytes
```

List of mutable types:

```
list, dict, set, bytearray, user-defined classes
```

How was the Python course by now? Did you satisfied?

Conditional Statements

Control Flow Statements

- Conditionals
- Loops

Table of Contents

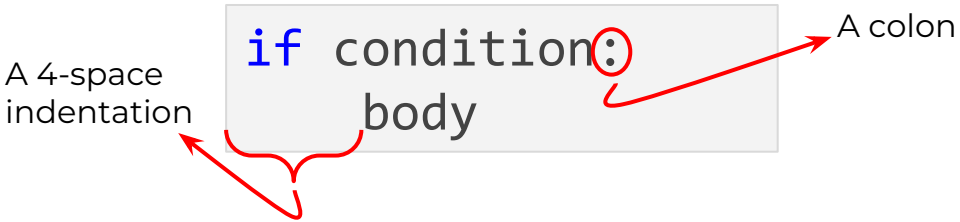
1

Structure of the `if` Statements

```
if a > b :
    print('a is greater than b')
```

Structure of the `if` Statements

- The basic structure 👇 of an `if` statement is :



Structure of the `if` Statements

- Here's the simple **pre-class** examples of the `if` Statements :

```
1  if True:
2      print('it is true')
3
```

Structure of the `if` Statements

- Here's the simple **pre-class** examples of the `if` Statements :

```
1  if True:
2      print('it is true')
3
```

```
1  it is true
2
```

`if` Statements

- **Task** : Cooking a **ham**burger.
  - We need some ingredients that are not in our kitchen.
  - There is only one **grocery store** in our village and its availability is crucial.
  - Ingredients (stated below) required for cooking hamburgers with **greens** (it does not matter which one. **lettuce** / **pepper**)
  - Set a **logical boolean algorithm** onto **hamburger** to be able to eat.
  - Set a condition `hamburger` variable with if statement that gives us a message "**Bon Appetit.**" if we can cook hamburger, do nothing if we can not.

```
#ingredients and requirements:
minced meat (must)
hamburger bread (must)
lettuce
pepper    } (must)
grocery store (must)
```

`if` Statements

- **The code can be like**:

```
1  minced = True
2  bread = True
3  lettuce = False
4  pepper = True
5  grocer = True
6
7  hamburger = (minced and grocer and bread) and (lettuce or pepper)
8
9  if hamburger :
10     print("Bon Appetit")
11
12
```

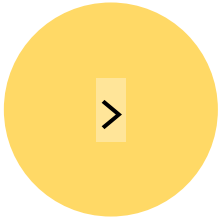
⚠️ The values (`True`/`False`) are up to you

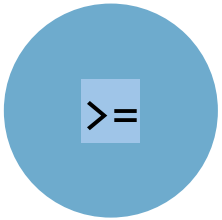
Output

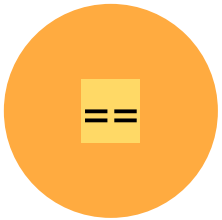
```
Bon Appetit
```

2 Comparison Operators

Draw lines to match the image to the answer:







equal

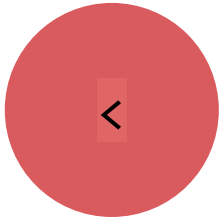
not equal

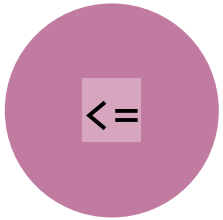
greater than

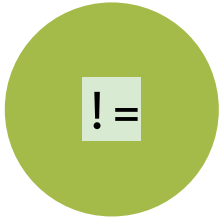
greater than or equal

less than

less than or equal







Students, draw anywhere on this slide!

Comparison Operators

These Operators return `True` or `False`

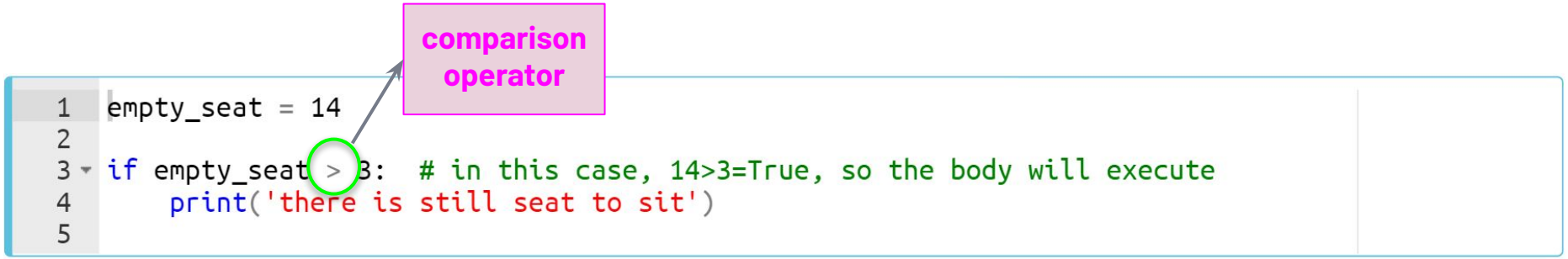
Comparison Operators

- Here's the simple **pre-class** examples of the `if` Statements :

```
1  empty_seat = 14
2
3  if empty_seat > 3:  # in this case, 14>3=True, so the body will execute
4      print('there is still seat to sit')
5
```

Comparison Operators

- Here's the simple **pre-class** examples of the `if` Statements :




```
there is still seat to sit
```

Comparison Operators

- Take a look at the following examples :

```
1  print(1 == 1)
2  print("henry" == "Henry")
3  print(12 < 12.1)
4  print("hard" != "easy")
5
6
```

Output

```
True
False
True
True
```

Comparison Operators (review)

Opr.	How it works ?	Sample
`==`	Returns `True` if two values are equal or `False` if different	2 `==` 2 (`True`), 2 `==` 3 (`False`)
`!=`	Returns `True` if two values are not equal or `False` if equal	2 `!=` 2 (`False`), 2 `!=` 3 (`True`)
`>`	Returns `True` if the value on the left is greater than the value on the right otherwise returns `False`	3 `>` 2 (`True`), 2 `>` 3 (`False`)
`<`	Returns `True` if the value on the left is less than the value on the right otherwise returns `False`	2 `<` 3 (`True`), 3 `<` 2 (`False`)
`>=`	Returns `True` if the value on the left is greater than or equal to the value on the right otherwise returns `False`	3 `>=` 2 (`True`), 3 `>=` 3 (`True`), 2 `>=` 3 (`False`)
`<=`	Returns `True` if the value on the left is less than or equal to the value on the right otherwise returns `False`	3 `<=` 2 (`False`), 3 `<=` 3 (`True`), 2 `<=` 3 (`True`)

Comparison Operators

- Let's examine the following **pre-class** example carefully :

```
1  x = 6
2  y = 9
3  print ("is x equal to y?                 :" , x == y)
4  print ("is x not equal to y?             :" , x != y)
5  print ("is x less than y?                :" , x < y)
6  print ("is x greater than y?             :" , x > y)
7  print ("is x less than or equal to y?    :" , x <= y)
8  print ("is x greater than or equal to y? :" , x >= y)
9
```

Comparison Operators

- Let's examine the following example carefully :

```
1  x = 6
2  y = 9
3  print ("is x equal to y?                :" , x == y)
4  print ("is x not equal to y?            :" , x != y)
5  print ("is x less than y?               :" , x < y)
6  print ("is x greater than y?            :" , x > y)
7  print ("is x less than or equal to y?   :" , x <= y)
8  print ("is x greater than or equal to y? :" , x >= y)
9
```

```
1  is x equal to y?                : False
2  is x not equal to y?            : True
3  is x less than y??              : True
4  is x greater than y?            : False
5  is x less than or equal to y?   : True
6  is x greater than or equal to y? : False
7
```

Comparison Operators

- **Task** :
  - Create two `sets` (using `set()` function) with the given `string` values below.
  - Compare these `sets` and print out '**We are the same!**' if they are equal, do nothing if they are not.

```
-  "TWELVE PLUS ONE"
-  "ELEVEN PLUS TWO"
```

Comparison Operators

- **The code might be like** :

```
1  set1 = set("TWELVE PLUS ONE")
2  set2 = set("ELEVEN PLUS TWO")
3
4  if set1 == set2:
5      print("We are the same!")
6
7
```

Discuss in-class! How they can be the same.

Output

```
We are the same!
```

Comparison Operators

- **Task** : **Convert string `"Yes"` to boolean `True`, convert string `"No"` to boolean `False`.**
  - Write a program that ;
    - Takes the word **Yes** or **No** from the users and **converts** it into **boolean** type.
    - Yes ➔ `True`
    - No (or other than Yes) ➔ `False`
    - Print the result i.e.(if *yes*): `"You entered True"`
    - You may not use if-statements

Comparison Operators

- **The code might be like** :

```
1  convert = input("Enter Yes or No : ").title().strip() == "Yes"
2  print("You entered", convert)
3
```

3 `if-else` Statements

`if-else` Statements(review)

- The simple structure 👇 of an `if-else` statement is :

```
if condition1:
    execute body1
else:
    execute body2
```

`if-else` Statements(review)

- Let's take a look at this **pre-class** example of an `if-else` statement :

```
1  course = 'clarusway'
2
3  if course == "clarusway":
4      print("you guaranteed the job")
5  else:
6      print("think about it again")
7
```

`if-else` Statements

- Let's take a look at this example of an `if-else` statement :

```
course = 'clarusway'

if course == "clarusway":
    print("you guaranteed the job")
else:
    print("think about it again")

```

```
you guaranteed the job

```

`if-else` Statements(review)

- Here's another **pre-class** example of an `if-else` statement :

```
1  number = 5
2  if number <= 3:
3      print("Number is smaller than or equal to 3")
4  else:  # Optional clause (you can only have one else)
5      print("Number is bigger than 3")
6
```

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

Students, write your response!

`if-else` Statements

- Here's another example of an `if-else` statement :

```
number = 5
if number <= 3:
    print("Number is smaller than or equal to 3")
else:  # Optional clause (you can only have one else)
    print("Number is bigger than 3")

```

```
Number is bigger than 3

```

`if-else` Statements

- **Task** : **Python Program to Check if a Number is Odd or Even**
  - Write a program to check whether a number entered by the user is **even** or **odd**.
  - Print the result such as : "**2 is even**"

`if-else` Statements

- **The code might be like** :

```
1  num = int(input("Enter a number: "))
2  if (num % 2) == 0:
3     print("{0} is Even".format(num))
4  else:
5     print("{0} is Odd".format(num))
6
7
```

`if-else` Statements

- **Task** : **Python Program to Check if a Number is Negative or Positive.**
  - Write a program to check whether a number entered by the user is *negative* or *positive*. Number zero is not acceptable.
  - Print the result such as : '**Positive number**”

`if-else` Statements

- **The code might be like** :

```
1  num = float(input("Enter a number: "))
2  if num > 0:
3      print("Positive number")
4  else:
5      print("Negative number")
6
7
```

`if-else` Statements

- **Task** : **Python Program to Check which number is larger.**
  - Write a program that prints which of the two numbers the user entered is large.
  - Print the result such as : "**The large number is 4**"

`if-else` Statements

- **The code might be like** :

```
1  num1 = float(input("Enter first number: "))
2  num2 = float(input("Enter second number: "))
3
4  if (num1 > num2) :
5     larger = num1
6  else:
7     larger = num2
8
9  print("The large number is", larger)
10
11
```

Option-1

`if-else` Statements

- **The code might be like** :

Option-1

```
1  num1 = float(input("Enter first number: "))
2  num2 = float(input("Enter second number: "))
3
4  if (num1 > num2) :
5     larger = num1
6  else:
7     larger = num2
8
9  print("The large number is", larger)
10
```

Option-2

```
1  num1 = float(input("Enter first number: "))
2  num2 = float(input("Enter second number: "))
3
4  if (num1 > num2) :
5     print("The large number is", num1)
6  else:
7     print("The large number is", num2)
8
```

`if-else` Statements

- **Task** : **Convert boolean `True` to string value of `"Yes"`, convert boolean `False` to string value of `"No"`.**
  - Write a program that ;
    - Converts the type of the variable `bool_value` which keeps `True` / `False` to **Yes** or **No**.
    - `True` → "Yes"
    - `False` → "No"

`if-else` Statements

- **The code might be like** :

```
1  bool_value = False  # can be True or False
2
3  if bool_value:
4      print("Yes")
5  else :
6      print("No")
7
```

Output

```
No
```

4 `if-elif-else` Statements

What do you know about statements:

`if-elif-else`

Students, write your response!

`if-elif-else` Statements (review)

- You can define a series of conditionals.

- `if` for the **first** one,
- `elif` for the **rest**, up until the final (optional),
- `else` for **anything not caught by the other conditionals.**

`if-elif-else` Statements (review)

- The simple and common structures of an `if-elif-else` statement are:

```
if condition1:
    execute body1

elif condition2:
    execute body2

else:
    execute body3
```

```
if condition_1:
    action_1

elif condition_2:
    action_2
    .
    .
    .
elif condition_n:
    action_n

else:
    action_(n+1)
```

here you can add as many elifs as you need

`if-elif-else` Statements (review)

- Consider this **pre-class** example :

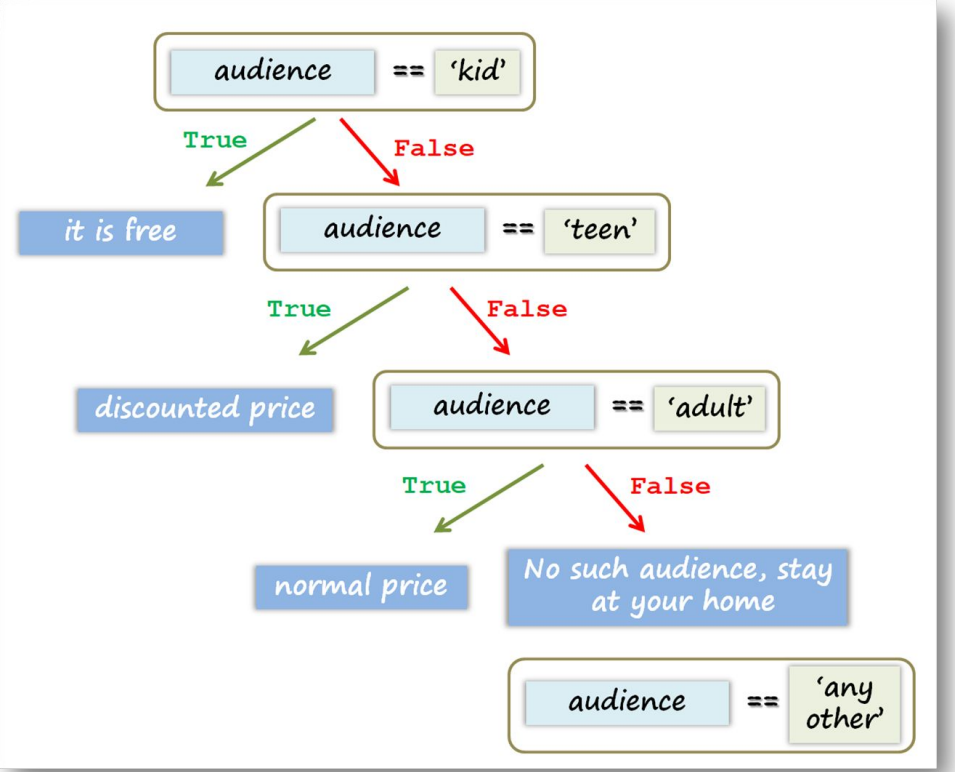
```
1  audience = "baby"
2
3  if audience == "kid":
4      print("it is free to go to cinema")
5  elif audience == "teen":
6      print("discounted price!")
7  elif audience == "adult":
8      print("normal price")
9  else:
10     print("No such audience, stay at your home!")
11
```

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

Students, write your response!

`if-elif-else` Statements (review)

- Let's examine this diagram of previous example :



`if-elif-else` Statements (review)

- The output :

```
1  audience = "baby"
2
3  if audience == "kid":
4      print("it is free to go to cinema")
5  elif audience == "teen":
6      print("discounted price!")
7  elif audience == "adult":
8      print("normal price")
9  else:
10     print("No such audience, stay at your home!")
11
```

```
1  No such audience, stay at your home!
2
```

`if-elif-else` Statements

- **Task : Write Python Program to Find the Largest Among Three Numbers**
  - Write a program that prints which of the three numbers the user entered is the largest.
  - Print the result such as : "**The largest number is 4**"

`if-elif-else` Statements

- **The code might be like** :

```
1  num1 = float(input("Enter first number: "))
2  num2 = float(input("Enter second number: "))
3  num3 = float(input("Enter third number: "))
4
5  if (num1 >= num2) and (num1 >= num3):
6     largest = num1
7  elif (num2 >= num1) and (num2 >= num3):
8     largest = num2
9  else:
10    largest = num3
11
12 print("The largest number is", largest)
13
```

`if-elif-else` Statements

- **Task** : **Write Python Program to Check if a Number is Negative, Positive or Zero.**
  - Write a program to check whether a number entered by the user is negative, positive or zero.
  - Print the result such as : “**Negative number**” or “**Zero**”.

`if-elif-else` Statements

- **The code might be like** :

```
1  num = float(input("Enter a number: "))
2  if num > 0:
3      print("Positive number")
4  elif num == 0:
5      print("Zero")
6  else:
7      print("Negative number")
8
```

5 Nested `if-elif-else` Statements

Nested `if-elif-else` Statements

- Nested structure of **pre-class** examples.

```
1  audience_group = 'kid', 'teen', 'adult'
2
3  audience = "teen"
4
5  if audience in audience_group:
6      if audience == "kid":
7          print("it is free to go to cinema")
8      elif audience == "teen":
9          print("discounted price!")
10     else: # audience == "adult":
11         print("normal price")
12 else:
13     print("No such audience, stay at your home!")
14
```

Nested `if-elif-else` Statements

- In this case, the output is :

```
1  audience_group = 'kid', 'teen', 'adult'
2
3  audience = "teen"
4
5  if audience in audience_group:
6      if audience == "kid":
7          print("it is free to go to cinema")
8      elif audience == "teen":
9          print("discounted price!")
10     else: # audience == "adult":
11         print("normal price")
12 else:
13     print("No such audience, stay at your home!")
14
```

```
1  discounted price!
2
```

Nested `if-elif-else` Statements

- Let's write a program that asks you to enter your exam score and calculates the range in which your degree is based on your exam score. The output would be: e.g, "**Your degree is B+**"
  - 95 and above ▸▸ "A+"
  - 90-94 ▸▸ "A"
  - 85-89 ▸▸ "B+"
  - 80-84 ▸▸ "B"
  - 79 and below ▸▸ "**below B**" or "B-"

⚠️ Use nested `if`-statement.

Nested `if-elif-else` Statements

- The one of the solution code may be like :

```
score = int (input("Enter your score :"))

if score >= 90:
    if score >= 95:
        Score_letter="A+"
    else:
        Score_letter="A"
elif score >= 80:
    if score >= 85:
        Score_letter="B+"
    else:
        Score_letter="B"
else:
    Score_letter="below B"

print ("Your degree: %s" % Score_letter)

```

THANKS!

End of the Lesson

(Conditional Statements)

next Lesson

Loops

click above