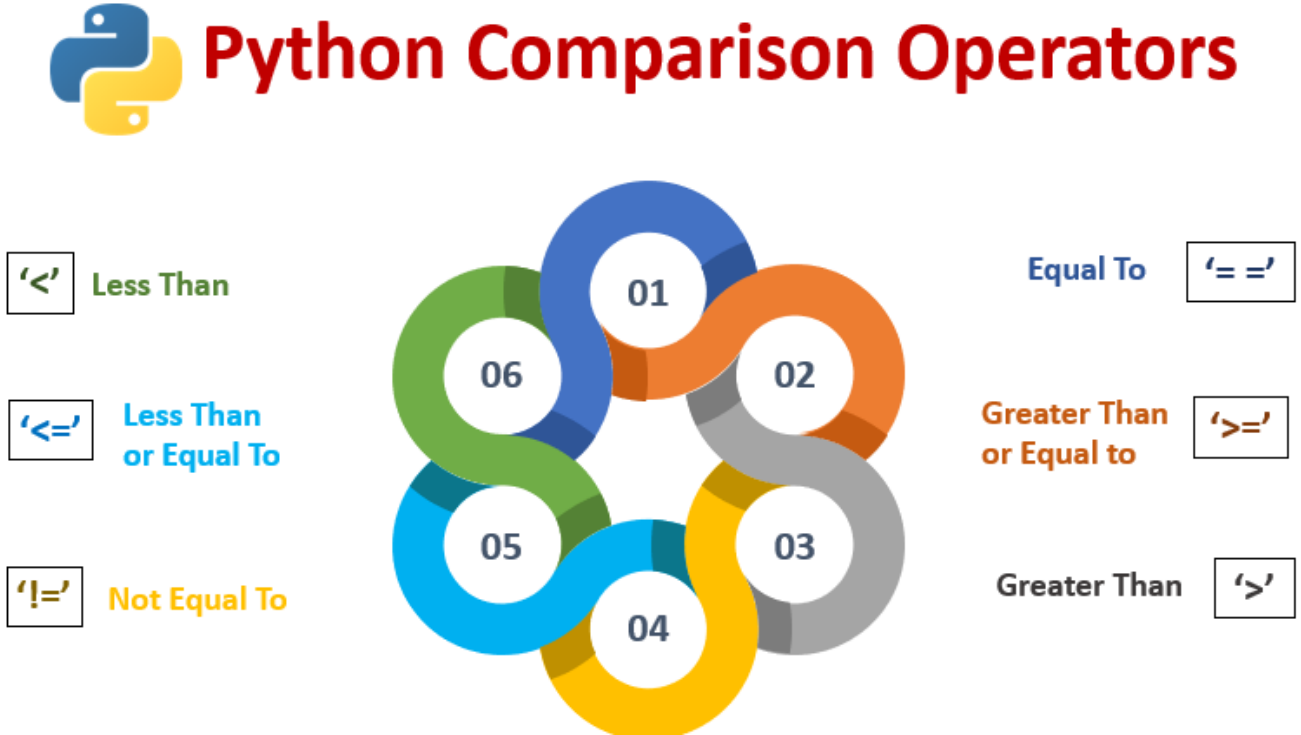
CHAPTER – 5

COMPARISION OPERATORS

COMPARISON OPERATORS

A comparison operator in python, also called python relational operator, compares the values of two operands and returns True or False based on whether the condition is met.



EQUALITY (==)

- To check if two Python values, or variables, are equal you can use ==.
- To check for inequality, you need !=.
- The equality operators will return True or False values.
- As a refresher, have a look at the following examples that all result in True.

Examples of equality operators

Data Type	Example
Integer	2 == (1 + 1)
String	"intermediate" != "python"
Boolean	True != False

When you write these comparisons in a script, you will need to wrap a print() function around them to see the output.

TASK

Write Python code to check if -5 * 15 is not equal to 75.

ANSWER

Comparison of integers

-5*15!=75

TASK

Check whether the strings "pyscript" and "PyScript" are equal.

ANSWER

```
# Comparison of strings
"pyscript"=="PyScript"
```

TASK

What happens if you compare booleans and integers? Write code to see if True and 1 are equal.

ANSWER

```
# Compare a boolean with an integer
True==1
```

GREATER THAN, LESS THAN

- You can use the less than and greater than signs, < and > in Python.
- You can combine them with an equals sign: <= and >=.
- Pay attention: <= is valid syntax, but =< is not.

All Python expressions in the following code chunk evaluate to True

Less than:

```
3 < 4
```

Less than equals to:

```
3 <= 4
```

Less than equals to on Strings:

```
"alpha" <= "beta"
```

For string comparison, Python determines the relationship based on alphabetical order.

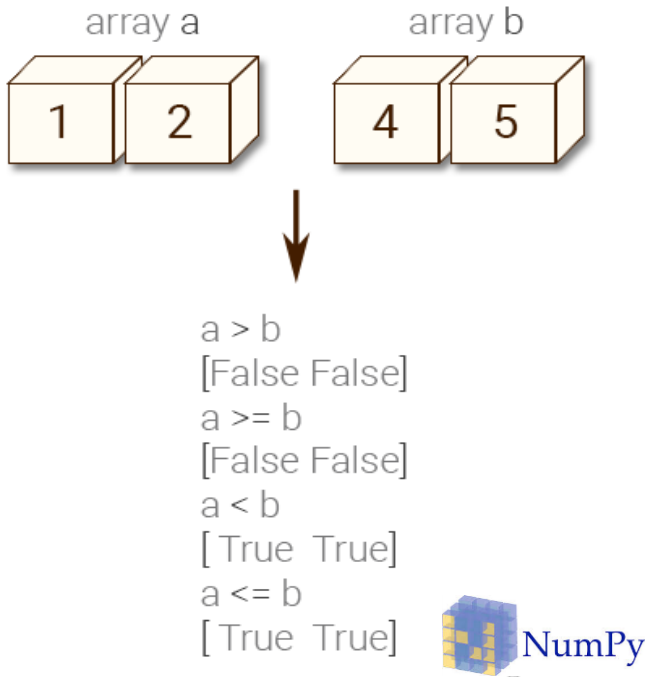
COMPARING NUMPY ARRAYS

Out of the box, you can also use comparison operators with Numpy arrays.

Example:

```
import numpy as np
a = np.array([1, 2])
b = np.array([4, 5])
```

Condition	Output
print(np.greater(a, b))	[False False]
print(np.greater_equal(a, b))	[False False]
print(np.less(a, b))	[True True]
print(np.less_equal(a, b))	[True True]



TASK

Create a numpy array my_house & your_house with areas for the kitchen, living room, bedroom and bathroom in the same order in the area list provided below.

```
#area list
my_house = (18.0, 20.0, 10.75, 9.50)
your_house = (14.0, 24.0, 14.25, 9.0)
```

ANSWER

```
# Creating arrays
import numpy as np
my_house = np.array([18.0, 20.0, 10.75, 9.50])
your_house = np.array([14.0, 24.0, 14.25, 9.0])
```

TASK

Using comparison operators, generate boolean arrays to answer the following question:

Which areas in my_house are greater than or equal to 18?

ANSWER

```
# Create arrays
import numpy as np
my_house = np.array([18.0, 20.0, 10.75, 9.50])
# my_house greater than or equal to 18
print(my_house >=18)
```

TASK

Using comparison operators, generate boolean arrays to answer the following question:

Which areas in your_house is less than or equal to 15?

ANSWER

```
# Create arrays
import numpy as np
your_house = np.array([14.0, 24.0, 14.25, 9.0])
# my_house less than or equal to 15
print(my_house <=15)
```

TASK

Compare two Numpy arrays and print which areas in my_house are smaller than the ones in your_house?

ANSWER

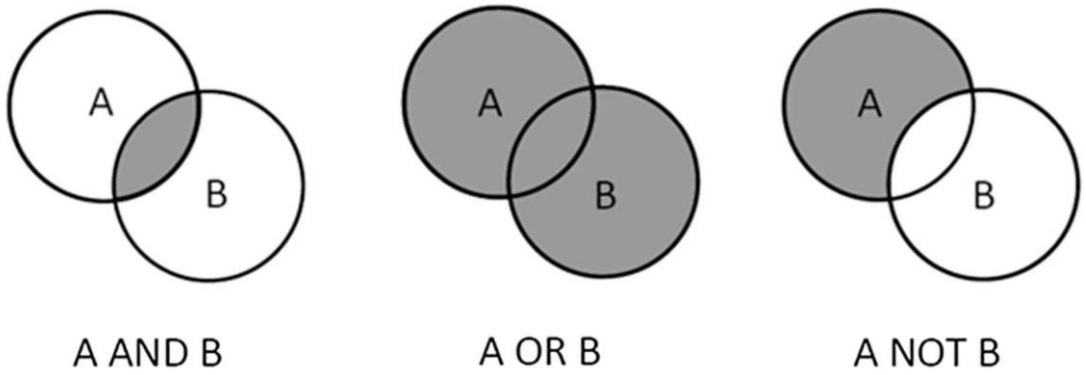
```
# Create arrays
import numpy as np
my_house = np.array([18.0, 20.0, 10.75, 9.50])
your_house = np.array([14.0, 24.0, 14.25, 9.0])
# my_house less than your_house
print(my_house<your_house)
```

BOOLEAN OPERATORS

The logical operators and, or and not are also referred to as boolean operators. While and as well as or operator needs two operands, which may evaluate to true or false,

Not operator needs one operand evaluating to true or false

The AND operator restricts the number of hits
The OR operator expands the number of hits
The NOT operator also restricts the number of hits



AND, OR, NOT

A boolean is either 1 or 0, True or False.

With boolean operators such as and, or and not, you can combine these booleans to perform more advanced queries on your data.

In the sample code below, two variables are defined: my_kitchen and your_kitchen, representing areas.

```
# Define variables
my_kitchen = 18.0
your_kitchen = 14.0
```

TASK

Write Python expressions, wrapped in a print() function, to check whether my_kitchen is bigger than 10 and smaller than 18.

ANSWER

```
# Define variables
my_kitchen = 18.0
your_kitchen = 14.0
# my_kitchen bigger than 10 and smaller than 18?
print(my_kitchen>10 and my_kitchen<18 )
```

TASK

Write Python expressions, wrapped in a print() function, to check whether my_kitchen is smaller than 14 or bigger than 17.

ANSWER

```
# Define variables
my_kitchen = 18.0
your_kitchen = 14.0
# my_kitchen smaller than 14 or bigger than 17?
print(my_kitchen<14 or my_kitchen>17 )
```

TASK

Write Python expressions, wrapped in a print() function, to check whether: double the area of my_kitchen is smaller than triple the area of your_kitchen.

ANSWER

```
# Define variables
my_kitchen = 18.0
your_kitchen = 14.0
#Double my_kitchen smaller than triple your_kitchen?
print(my_kitchen*2< your_kitchen*3)
```

BOOLEAN OPERATORS WITH NUMPY

Before, the operational operators like < and >= worked with Numpy arrays out of the box.

Unfortunately, this is not true for the boolean operators and, or, and not.

To use these operators with Numpy, you will need np.logical_and(), np.logical_or() and np.logical_not().

Here's an example on the my_house and your_house arrays from before to give you an idea:

Which areas in my house is greater than 13 and is less than 15 in your house.

```
np.logical_and(my_house > 13, your_house < 15)
```

TASK

Which areas in my_house are greater than 18.5 or smaller than 10?

ANSWER

```
# Create arrays
import numpy as np
my_house = np.array([18.0, 20.0, 10.75, 9.50])
# my_house greater than 18.5 or smaller than 10
print(np.logical_or(my_house>18.5,my_house<10))
```

TASK

Which areas are smaller than 11 in both my_house and your_house?.

ANSWER

```
# Create arrays
import numpy as np
my_house = np.array([18.0, 20.0, 10.75, 9.50])
your_house = np.array([14.0, 24.0, 14.25, 9.0])
# Both my_house and your_house smaller than 11
print(np.logical_and(my_house<11,your_house<11))
```

Conditional Statements

CONDITIONAL STATEMENT - IF

Conditional Statement in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false.

Conditional statements are handled by IF statements in Python.

IF

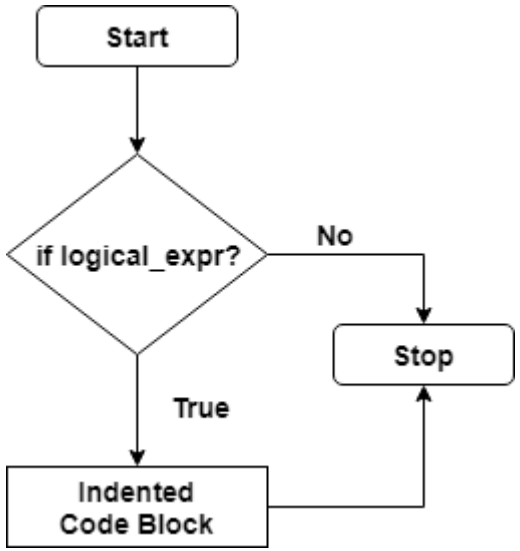
In Python, If Statement is used for decision making.

It will run the body of code only when IF statement is true.

When you want to justify one condition while the other condition is not true, then you use "if statement".

Syntax: if Logical_Expression :

Indented Code Block



IF-EXAMPLE

To experiment with if have a look at this code sample:

```
#defining area
area = 10.0
#using if statement
if(area < 9) :
    print("small")
print("large")
```

out:

large

TASK- BACKGROUND

Two variables are defined in the sample code: room, a string that tells you which room of the house we're looking at, and area, the area of that room.

```
# Define variables
room = "kit“
area = 14.0
```

TASK

Use an if statement that prints out "Looking around in the kitchen." if room equals "kit".

ANSWER

```
# Define variables
room = "kit"
# if statement for room
if room == "kit" :
  print("looking around in the kitchen.")
```

TASK

Write another if statement that prints out "big place!" if area is greater than 15.

ANSWER

```
# Define variables
area = 14.0
# if statement for area
if area > 15:
  print("big place!")
```

ADD ELSE

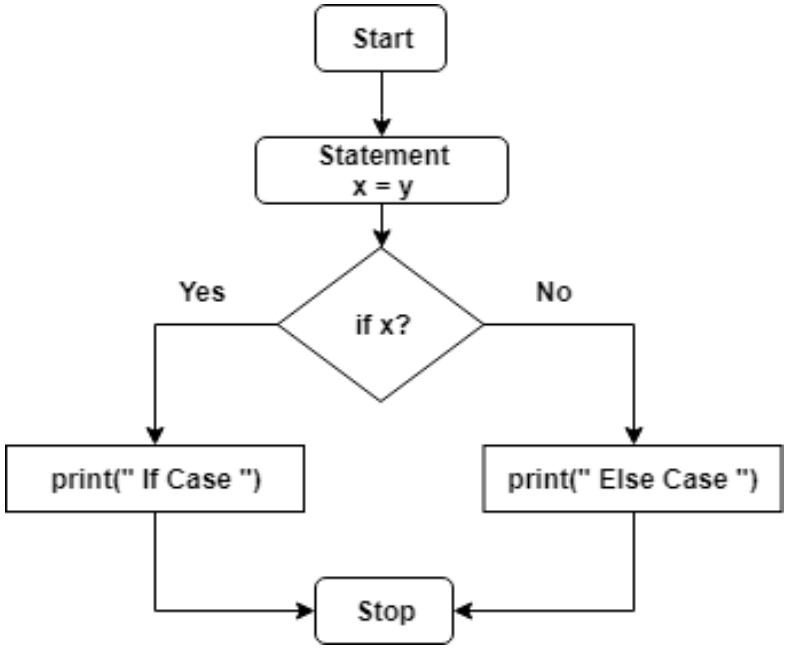
On the right, the if construct for room has been extended with an else statement so that "looking around elsewhere." is printed if the condition room == "kit" evaluates to False.

IF ELSE

- A Python if else statement takes action irrespective of what the value of the expression is.
- If the result is True, then the code block following the expression would run. Otherwise, the code indented under the else clause would execute.

Syntax:

```
if Logical_Expression :
        Indented Code Block 1
else :
        Indented Code Block 2
```



IF ELSE

Below, the if construct for room has been extended with an else statement so that "looking around elsewhere." is printed if the condition room == "kit" evaluates to False.

```
# Define variables
room = "kit“
area = 14.0

# if-else construct for room
if room == "kit" :
    print("looking around in the kitchen.")
else :
    print("looking around elsewhere.")
```

Out: looking around in the kitchen.

TASK

Add an else statement so that "pretty small." is printed out if area > 15 evaluates to False.

```
# if-else construct for area
if area > 15 :
    print("big place!")
```

ANSWER

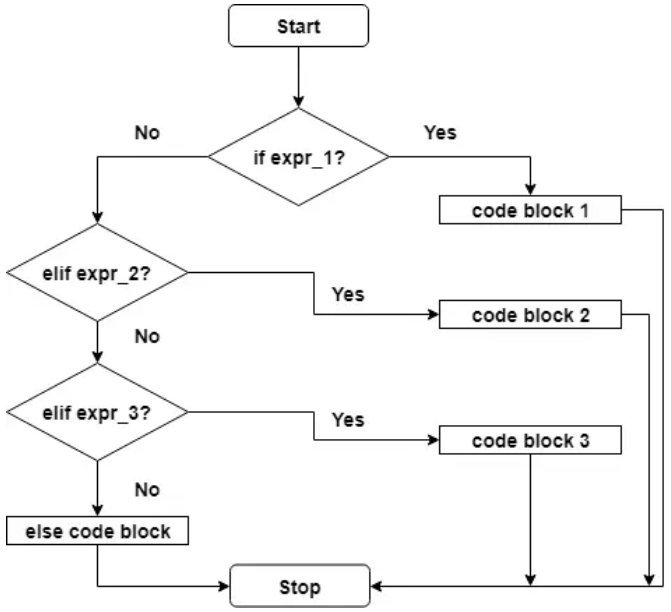
```
# Define variables
area = 14.0
# if statement for area
if area > 15:
    print("big place!")
else :
    print("pretty small.")
```

ELIF

- The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".
- If the ELIF result is True, then the code block following the expression would run.

Syntax:

```
if Logical_Expression_1 :
        Indented Code Block 1
elif Logical_Expression_2 :
        Indented Code Block 2
elif Logical_Expression_3 :
        Indented Code Block 3
...
else :
        Indented Code Block N
```



CUSTOMIZE FURTHER: ELIF

It's also possible to have a look around in the bedroom.

The sample code contains an elif part that checks if room equals "bed". In that case, "looking around in the bedroom." is printed out.

```
# Define variables
room = "bed"
# if-elif-else construct for room
if room == "kit" :
    print("looking around in the kitchen.")
elif room == "bed":
    print("looking around in the bedroom.")
else :
    print("looking around elsewhere.")
```

TASK

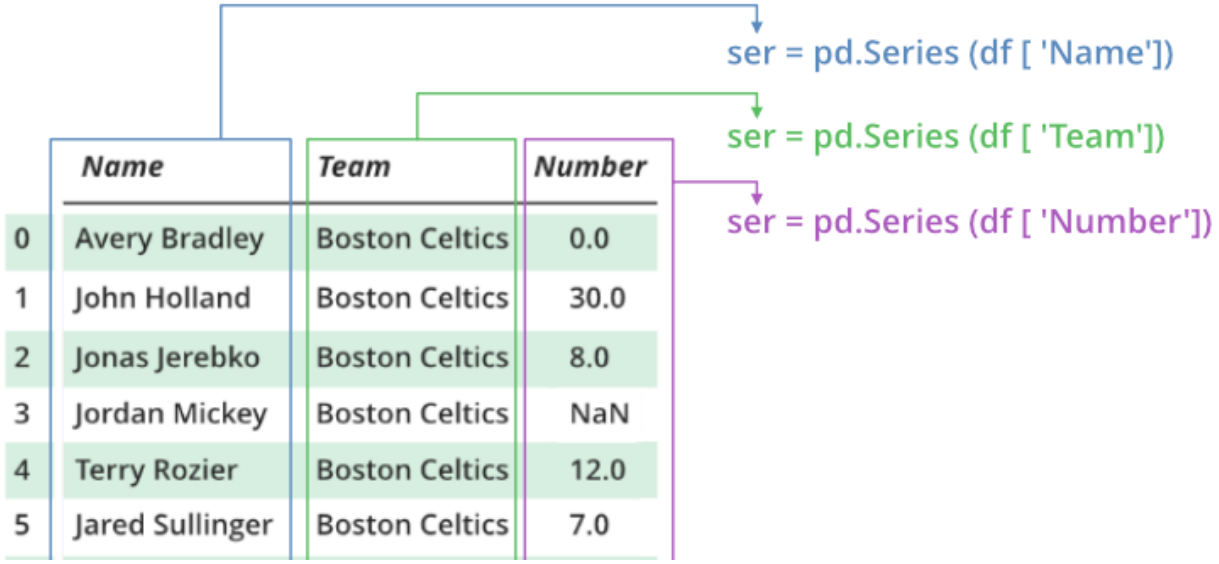
Add an elif statement so that "medium size, nice!" is printed out if area is greater than 10.

ANSWER

```
# Define variables
area = 14.0
# if statement for area
if area > 15:
    print("big place!")
elif area >10:
    print("medium size, nice!")
```

PANDAS SERIES

- Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float).
- The axis labels are collectively called index.
- In the real world, a Pandas Series will be created by loading the datasets from existing storage such as SQL Database, CSV file, or an Excel file.
- Pandas Series can be created from the lists or dictionaries.



	Name	Team	Number
0	Avery Bradley	Boston Celtics	0.0
1	John Holland	Boston Celtics	30.0
2	Jonas Jerebko	Boston Celtics	8.0
3	Jordan Mickey	Boston Celtics	NaN
4	Terry Rozier	Boston Celtics	12.0
5	Jared Sullinger	Boston Celtics	7.0

PANDAS SERIES-EXAMPLE

- Creating a series from array: In order to create a series from array, we have to import a numpy module and have to use array() function.

Example creating Pandas series from an array "geeks"

```
# import pandas as pd
import pandas as pd
# import numpy as np
import numpy as np
# simple array
data = np.array(['g','e','e','k','s'])
# Creating a series from array
ser = pd.Series(data)
print(ser)
```

Out:

```
0    g
1    e
2    e
3    k
4    s
```

TASK

Create pandas pd_ser series using the given array and print out type of pd_ser.

ser=['g','e','e','k','s','f','o','r','g','e','e','k','s']

ANSWER

```
# import pandas and numpy
import pandas as pd
import numpy as np
# creating simple array
ser=['g','e','e','k','s','f', 'o','r','g','e','e','k','s']
# creating numpy array
data = np.array(ser)
# Creating a series from array
pd_ser = pd.Series(data)
#print pandas series
print(pd_ser)
#print type of pd_ser
print(type(pd_ser))
```

ACCESS SERIES - FROM DATAFRAME

We can access series from dataframe pandas function for Series.

Syntax: pd.Series(df['columeName'])

```
# importing pandas module
import pandas as pd
# making data frame
df=pd.read_csv("https://raw.githubusercontent.com/Masadn/PythonCourse/master/dataset/nba.csv")
#access series from dataframe
ser = pd.Series(df['Name'])
print(ser)
```

Out:

```
0      Avery Bradley
1        Jae Crowder
2       John Holland
3        R.J. Hunter
4      Jonas Jerebko
           ...
453     Shelvin Mack
454        Raul Neto
455     Tibor Pleiss
456      Jeff Withey
457              NaN
```

DRIVING RIGHT – TASK BACKGROUND

Remember that cars dataset, containing the cars per 1000 people (cars_per_cap) and whether people drive right (drives_right) for different countries (country)? The code that imports this data in CSV format into Python as a DataFrame is available.

```
# Import cars data
import pandas as pd
cars = pd.read_csv('https://raw.githubusercontent.com/Masadn/PythonCourse/master/dataset/cars.csv', index_col = 0)
```

TASK

Extract the drives_right column as a Pandas Series and store it as a variable dr.

Use dr, to subset the cars DataFrame and store the resulting selection in a variable sel.

Print sel, and assert that drives_right is True for all rows.

ANSWER

```
# Import cars data
import pandas as pd
cars = pd.read_csv('https://raw.githubusercontent.com/Masadn/PythonCourse/master/dataset/cars.csv', index_col = 0)
print(cars)
# Extract drives_right column as Series: dr
dr=pd.Series(cars['drives_right'])
print(dr)
# Use dr to subset cars: sel
sel=cars[dr]
# Print sel
print(sel)
```

Q&A

python

THANK YOU