Lecture 2: Programming Languages Categories

as we had discussed before, depending on the evolution path that each real life application needed from the programming language it used, we got several programming languages categories, each having its own advantages and disadvantages.

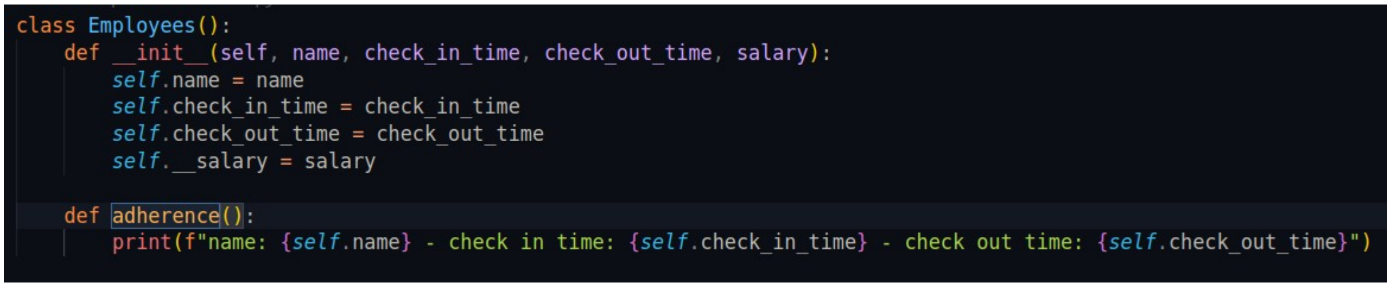
In the following we will be studying some of these categories to see what are the application fields for each, benefits, and used.

- ## OOP (Object Oriented Programming):

  let’s make a check in system for a company that tracks the attendance for the employees. In this case we will have different types of employees with different check in times, which will lead to the need of classifying them into separate groups, each group will have some distinct features from the other, and some groups might share some features with other ones.
  For example, we can make a group for low level employees, and another one for management employees, and another for the owners of the company. The group of the low level employees will have smaller groups inside of it such as secretary, finance, security, … etc.
  From here on we will add different features (attributes) to each group. For example, the management group members will have the following attributes: name, role, check in time, check out time, salary, team members, department, and so on. Similarly, the other groups will have other information related to them and the nature of their work.

  this way of thinking is called Object Oriented Programming, where we separate our data into classes that share the same attributes, and each class can have sub-classes under it that have more specific attributes related to it specifically.

  Here is an example of the above in Python:

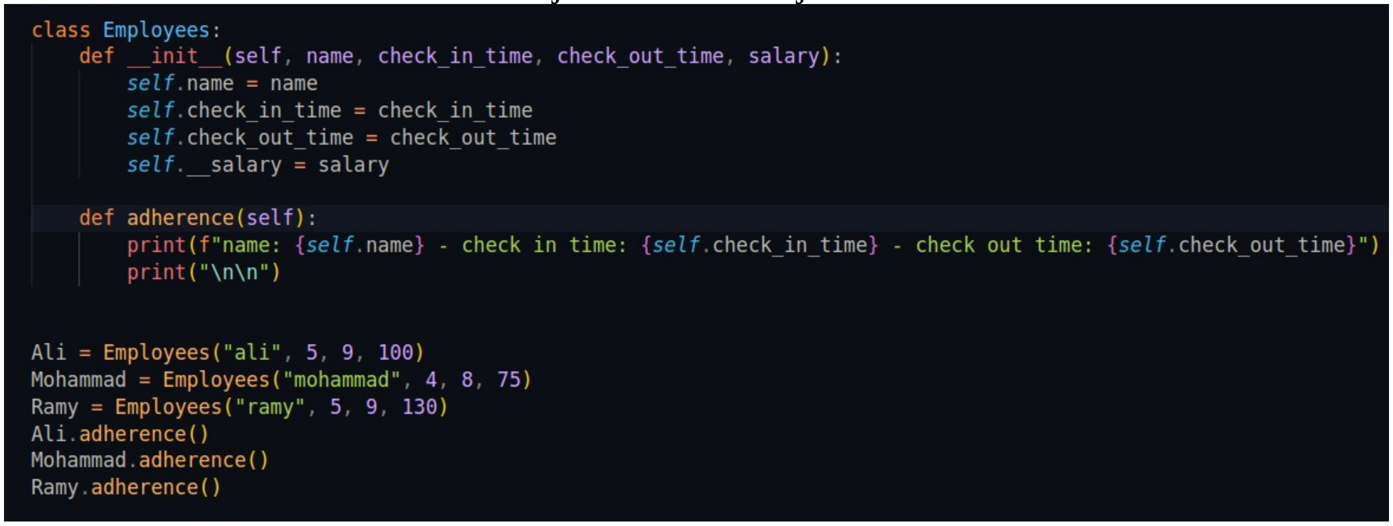


```python
class Employees():
    def __init__(self, name, check_in_time, check_out_time, salary):
        self.name = name
        self.check_in_time = check_in_time
        self.check_out_time = check_out_time
        self.__salary = salary

    def adherence():
        print(f"name: {self.name} - check in time: {self.check_in_time} - check out time: {self.check_out_time}")
```

  As we can see, we classified the Employees as a separate *object* (we will be calling them objects as the name implies, Object Oriented Programming), and

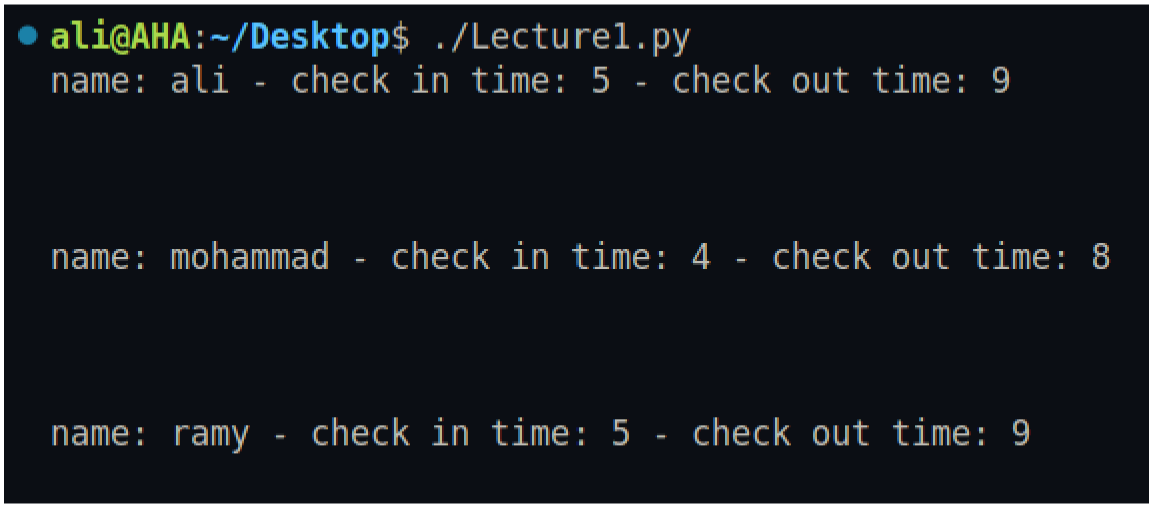
then gave the object its own *attributes*, which are the name, check in time, check out time, and the salary.
Then, we added a method to be able to interact with the object of Each employee to now what are the times the employee needs to be available at.

Here is how we can use the object that we had just created:



```python
class Employees:
    def __init__(self, name, check_in_time, check_out_time, salary):
        self.name = name
        self.check_in_time = check_in_time
        self.check_out_time = check_out_time
        self.__salary = salary

    def adherence(self):
        print(f"name: {self.name} - check in time: {self.check_in_time} - check out time: {self.check_out_time}")
        print("\n\n")


Ali = Employees("ali", 5, 9, 100)
Mohammad = Employees("mohammad", 4, 8, 75)
Ramy = Employees("ramy", 5, 9, 130)
Ali.adherence()
Mohammad.adherence()
Ramy.adherence()
```

We used the Employees class to create a new object with the data related to it, and then used the method that we assigned to it to be able to do an operation on these new objects that we created.
The output of the above program will be as follows:



```
ali@AHA:~/Desktop$ ./Lecture1.py
name: ali - check in time: 5 - check out time: 9



name: mohammad - check in time: 4 - check out time: 8



name: ramy - check in time: 5 - check out time: 9
```

As we can see, the “adherence()” method gave us the data that we needed to now about the times when each employee had to available at.

So in conclusion, in OOP languages, we will be mainly interested in encapsulating data with its relevant attributes and methods, to be able afterwards to do all the operations we need to do on them.

The way we implement this approach of programming may differ from one language to another depending on the syntax of the language, but at the end, the concept and the way we problem solve the tasks we have will be the same.

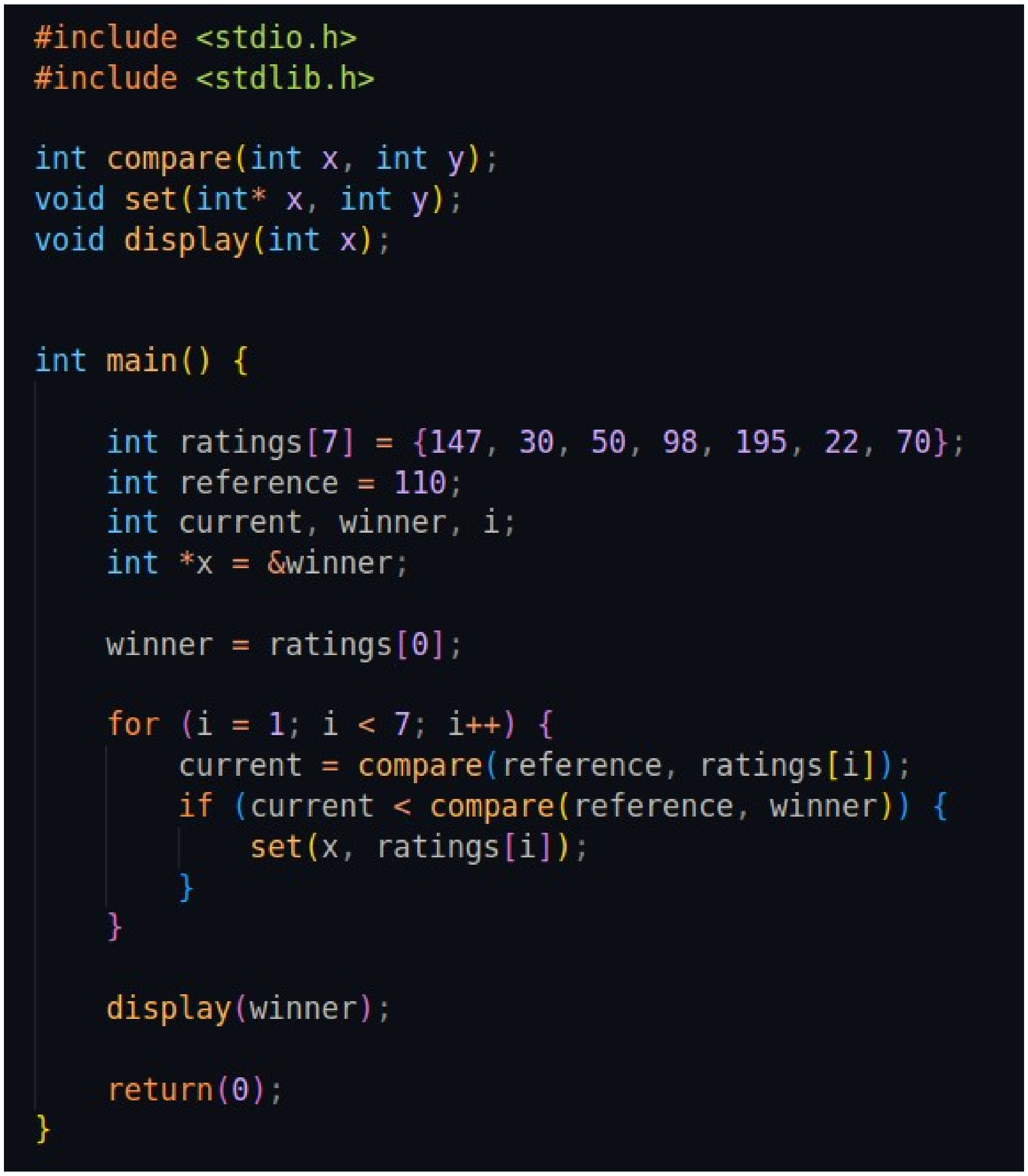
- Procedural Programming:

A client came to us with a set of data that represent the ratings of applicants to role in his company, and he wanted us to calculate for him the closest one to the rating that the company is requiring for the vacancy in the company.
To do so we will need to do a set of procedures on this set of data to be able to determine which of the applicants best fits the offered job.
Let's say the ratings were as follows: [147, 30, 50, 98, 195, 22, 70], and the required rating that the company needs is 110, so to be able to determine who has the closest rating we will need to go through the values one by one, compare it to the values we have, and then decide if it is close enough or not, which sums the procedures as follows:

1. go through the values
2. compare each value to the one we have to know how close it is to our desired value
3. decide if this is the closest one to us
4. conclude the final value and take it as an output

This way of breaking down the process into smaller procedures is what we call Procedural Programming, which aims to make the repetitive and big tasks, smaller and reusable over the whole project.

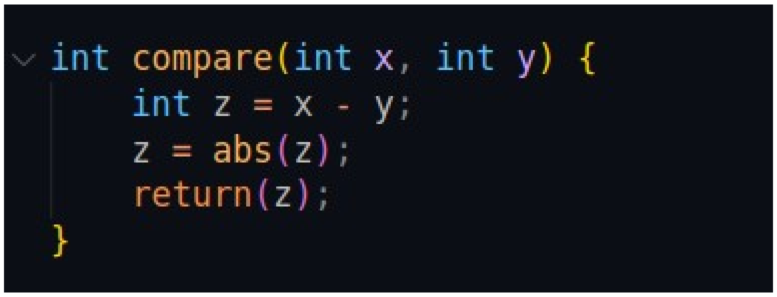
Here is how this will be done in C language:



```c
#include <stdio.h>
#include <stdlib.h>

int compare(int x, int y);
void set(int* x, int y);
void display(int x);


int main() {

    int ratings[7] = {147, 30, 50, 98, 195, 22, 70};
    int reference = 110;
    int current, winner, i;
    int *x = &winner;

    winner = ratings[0];

    for (i = 1; i < 7; i++) {
        current = compare(reference, ratings[i]);
        if (current < compare(reference, winner)) {
            set(x, ratings[i]);
        }
    }

    display(winner);

    return(0);
}
```

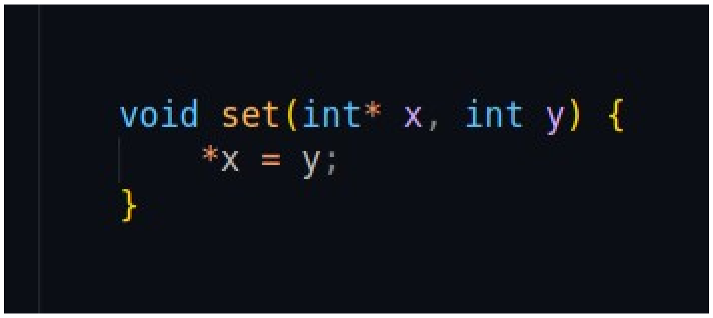
In the above code, we see how we had first declared the procedures we will be using in our task to be able to calculate the closest to the target. Then we implemented these procedures where we needed them in our code to get a clear and readable code that we can easily maintain in the future.
Here is how the code is working:

1. `int compare(int x, int y)`
   This procedure (later on we will be calling it “function”) takes two values, and returns the absolute difference between them, to know how close they are to each other.



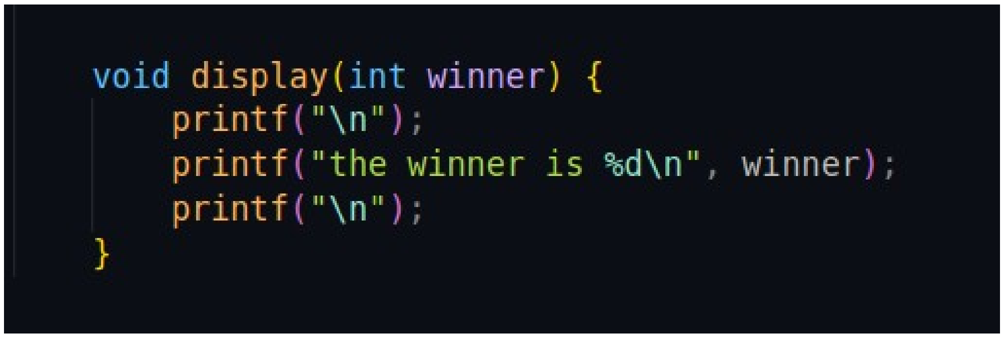
```c
int compare(int x, int y) {
    int z = x - y;
    z = abs(z);
    return(z);
}
```

2. `void set(int* x, int y)`
   This procedures operate inside a specific condition, where if the condition is satisfied, it takes the given value, and assigns it to the “winner” value.



```c
void set(int* x, int y) {
    *x = y;
}
```

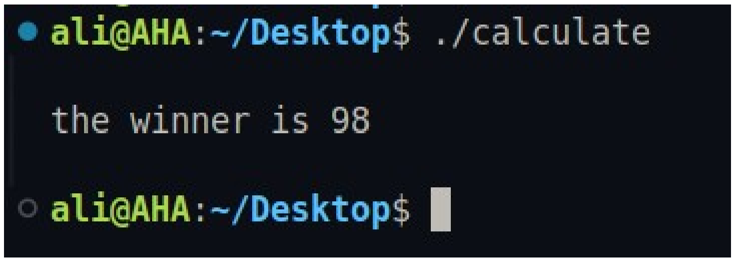
3. `void display(int x)`
   Lastly, we have the final procedure that takes the winner and prints it in a certain way to make it as pleasing as possible



```c
void display(int winner) {
    printf("\n");
    printf("the winner is %d\n", winner);
    printf("\n");
}
```

Note that the “for” loop and the “if” condition are in of themselves procedures too, used to control the execution flow of the program.

Here is the output of the above program:



```
ali@AHA:~/Desktop$ ./calculate

the winner is 98

ali@AHA:~/Desktop$
```

- Other programming languages categories exist and are used in every day-to-day application, but since they are out of the scope of our course, we will be covering them briefly in the following section:

    1. DSL (Domain Specific Language): this type of languages is made to serve the needs of a specific domain such as simulations (MATLAB), hardware description (VHDL), statistics (R), and more.

    2. Declarative languages: this type of languages is more concerned with telling the executer what to do as a task and not how to do it. A good example of this is SQL, where it is used to tell a database server what to send as data and what to do on this data before sending it, regardless of the way it will do it.
    Another example is HTML, where it tell the browser what it needs to show to the user without worrying about how the browser will do it.

Best wishes, Eng. Ali Hassan.