Build a python code and write a condition to detect an alarm

How do an Alarm Clock works?

An alarm clock usually requires you to set the specific time you want the alarm to ring. Once you have set your preferred time, the alarm will continuously match the time you provided with the current time. As soon as both the time matches, the alarm rings.

This is a very general idea of how usually an real alarm clock works. The alarm clock we will be building will follow the same mechanism.

Project Setup

Install required modules/libraries

Alright, so first things first!

In this project, we are going to use some external modules which are already made available by other developers. These modules will help us save a lot of time and effort. All we have to do is import them into our project to get started.

Importing modules is pretty simple. All you have to do is run a simple **pip install** command from terminal & our specified module will be downloaded in our system.

We need 2 different modules for our project - **datetime** & **playsound.**

Let's run pip install command and download both of these modules.

pip install datetime

datetime - We will use this module to obtain current time which is not possible without this module.

pip install playsound

playsound - We will use this module to play our alarm tone once the alarm rings.

Download alarm ringtone

datetime - We will use this module to obtain current time which is not possible without this module.

Build a python code and write a condition to detect an alarm

pip install playsound

playsound - We will use this module to play our alarm tone once the alarm rings.

Let's Code

So the first we are going to do is, of course, import both of our modules, we just installed.

```
from datetime import datetime
from playsound import playsound
```

Both of our modules are now ready to use.

Now let's ask the user for the time when the alarm will go off.

```
alarm_time = input("Enter time in 'HH:MM:SS AM/PM' format: ")
```

We need to have a pre-defined format in which the user will enter the time. Here we are using a standard time format **HH:MM:SS AM/PM** which asks for Hour, minute, second & period (AM/PM). We will save this input into **alarm_time** variable.

```
def validate_time(alarm_time):
    if len(alarm_time) != 11:
        return "Invalid time format! Please try again..."
    else:
        if int(alarm_time[0:2]) > 12:
            return "Invalid HOUR format! Please try again..."
        elif int(alarm_time[3:5]) > 59:
            return "Invalid MINUTE format! Please try again..."
        elif int(alarm_time[6:8]) > 59:
            return "Invalid SECOND format! Please try again..."
        else:
            return "ok"
```

Here is our function called **validate_time**. Let's break it down and understand what is going on -

- Our function accepts the user input as a parameter **alarm_time**.

Build a python code and write a condition to detect an alarm

- **In first if statement**, at **len(alarm_time) != 11** we are checking the length of **user input to be exactly 11 characters**. If not then it will **return** a statement, asking the user to re-enter the value. If the user input is exactly 11 characters long, then **else** block will execute, this is where the more in-depth validation of our user input happens.
- **In the first if statement within else block**, we are validating the first two characters of our input which are **HH.** There could be a slight chance that user may enter invalid hour values like something more than 12 hours. Here at **alarm_time[0:2],** we are using a slicing operator to access the first two characters of user input. The input is not more than 12 hours then the execution will move forward to the next conditional statement. But if the input is more than 12 hours, then it will **return** a statement asking the user to re-enter the time.
- Our function accepts the user input as a parameter **alarm_time**.
- **In first if statement**, at **len(alarm_time) != 11** we are checking the length of **user input to be exactly 11 characters**. If not then it will **return** a statement, asking the user to re-enter the value. If the user input is exactly 11 characters long, then **else** block will execute, this is where the more in-depth validation of our user input happens.
- **In the first if statement within else block**, we are validating the first two characters of our input which are **HH.** There could be a slight chance that user may enter invalid hour values like something more than 12 hours. Here at **alarm_time[0:2],** we are using a slicing operator to access the first two characters of user input. The input is not more than 12 hours then the execution will move forward to the next conditional statement. But if the input is more than 12 hours, then it will **return** a statement asking the user to re-enter the time.
- while True:
- alarm_time = input("Enter time in 'HH:MM:SS AM/PM' format: ")
-
- validate = validate_time(alarm_time.lower())
- if validate != "ok":
- print(validate)
- else:
- print(f"Setting alarm for {alarm_time}...")
- break
- Here we are storing the output of the function into a variable validate which we are using to judge whether the input is valid or not. If it is not valid then the user will be prompted to enter the time again. If not then the execution will head to the next step.

Build a python code and write a condition to detect an alarm

- Now we are sure that the input provided by the user is valid and now we can separately store the values into different variables. Have a look at the code.
- alarm_hour = alarm_time[0:2]
- alarm_min = alarm_time[3:5]
- alarm_sec = alarm_time[6:8]
- alarm_period = alarm_time[9:].upper()
- Here we are using slicing operator to store the specific unit of time into specific variables. **HH** will be stored in **alarm_hour**, **MM** in **alarm_min** and so on.
- now = datetime.now()
-
- current_hour = now.strftime("%I")
- current_min = now.strftime("%M")
- current_sec = now.strftime("%S")
- current_period = now.strftime("%p")
- if alarm_period == current_period:
- if alarm_hour == current_hour:
- if alarm_min == current_min:
- if alarm_sec == current_sec:
- print("Wake Up!")
- playsound('D:/Library/Documents/Projects/Coding/Beginner Python Projects/Alarm Clock/alarm.wav')
- while True:
- now = datetime.now()
-
- current_hour = now.strftime("%I")
- current_min = now.strftime("%M")
- current_sec = now.strftime("%S")
- current_period = now.strftime("%p")
-
- if alarm_period == current_period:
- if alarm_hour == current_hour:
- if alarm_min == current_min:
- if alarm_sec == current_sec:
- print("Wake Up!")
- playsound('D:/Library/Documents/Projects/Coding/Beginner Python Projects/Alarm Clock/alarm.wav')
- break

```
while True:
    now = datetime.now()
```

Build a python code and write a condition to detect an alarm

```
    current_hour = now.strftime("%I")
    current_min = now.strftime("%M")
    current_sec = now.strftime("%S")
    current_period = now.strftime("%p")

    if alarm_period == current_period:
        if alarm_hour == current_hour:
            if alarm_min == current_min:
                if alarm_sec == current_sec:
                    print("Wake Up!")
                    playsound('D:/Library/Documents/Projects/Coding/Beginner Python
Projects/Alarm Clock/alarm.wav')
                    break
```

/

Build a python code and write a condition to detect an alarm