Lesson Plan 9 | Form 2 | Fundamentals of Threading

Objective

Students will be introduced to threading: how it is implemented, typical use cases for it, and how it can be used in Python.

Warm-up

Draw three diagrams on the board:

1. Ubuntu desktop with two programs running (i.e., web browser and **nc**).
2. Facebook browser window with chat box open.
3. Video game with car driving down a road.

Give students a minute or two to think about how many things the computer is doing in each diagram.

Presentation

Start with discussion of the various things the computer is doing in each diagram:

1. We can go to different Internet sites while using **nc** to chat with friends.
2. We can talk to multiple Facebook friends at once.
3. As we drive the car in the video game, the scenery changes, the wheel turns, etc.

Ask students to remind everyone about the part of the computer that runs our programs (CPU). Put an example of a 32-bit / 64-bit instruction on the board, have everyone be reminded that the computer only works in binary, and only runs one instruction at a time.

Ask students to remind everyone about the software that runs the computer and is responsible for telling the CPU what to do (OS). Ask for ideas about how the OS allows multiple programs to run at the same time (switches them on and off CPU very quickly many times).

After a good or close-enough answer, draw a time sequence diagram with two sides: Program A and B. Show how the OS switches between each one, interleaving various instructions. An infinite *while* loop in one of them may also be useful for instruction purposes.

At this point, write a few bullet points up on the board as key details:

- the OS allows us to create *threads*
- a program can create many threads if it wants
- the OS decides when to run each thread
- threads are like a program’s children
- a thread runs a function
- we can use threads in Python using the **threading** library (import threading)

Show a program on the board with two infinite *while* loops, one after the other; using pseudocode, have the first listen for connections from others, while the second sends messages to others. Ask students whether the second loop will ever run (no).

Walk the students through creating a function (see last bullet point above) that we can move one of the *while* loops into. Then ensure they all understand that now both while loops will run at the same time. Note that exact use of the Python threading API should not be emphasized here, save that for next lesson.

If time allows, write a program on the board that starts two threads for two different functions, both of which change a variable. Ask them which one they think will get run first (impossible to know, the computer will decide, and may do so differently each time). Can then explain that threads can get complicated, so it’s best to keep them simple and light like their name implies.

Guided Practice

None.

Independent Practice

None

Closing

Remind students that threads allow us to do multiple things at once, and thus are helpful in allowing us to create applications like chatrooms, games, etc.

In Hindsight

02/06/2017: A few students got the concept immediately, the rest understood parts of it. Two students were quick to suggest that the OS switched programs on and off the CPU, which was great to hear. One said “multi-tasking,” which I assume is a term learned in ICT theory. It’s likely that more of them will understand how threads work once they are in the lab with a sample program in front of them.