WORKSHEET

A2.1 Stack frames and the call stack

Recursion and computational thinking · A level · OCR H446 2.2.1, AQA 7517 4.1.1.15 · about 20 min

BugBotLab

Name _____ Class _____ Date _____

What this lesson is about

Return addresses, parameters and local variables: what a subroutine call pushes, and what a return pops.

- What has to be remembered
- Pushing and popping frames
- Each call has its own locals
- Modelling the stack yourself
- When the stack runs out

Questions

6 marks in all

1. Which of these does a stack frame store for a subroutine call? [1 mark]

Tick every answer that is true.

- ☐ **A** The return address
- ☐ **B** The parameters
- ☐ **C** The local variables
- ☐ **D** The source code of the subroutine

2. What happens to the call stack when a subroutine returns? [1 mark]

- ☐ **A** The top frame is popped and execution continues at its return address
- ☐ **B** A new frame is pushed
- ☐ **C** The bottom frame is popped
- ☐ **D** Every frame is cleared

3. Why is a stack the right structure for subroutine calls? [1 mark]

- ☐ **A** The most recently called subroutine is always the first to finish
- ☐ **B** Subroutines finish in the order they were called
- ☐ **C** It lets any frame be read directly by its position
- ☐ **D** It uses no memory

4. What does this program print? [1 mark]

```
def inner():
    total = 100
    return total

def outer():
    total = 5
    inner()
    return total

print(outer())
```

5. main calls a(), a() calls b(), and b() calls c(). Counting a frame for the main program, how many frames are on the call stack while c() runs? [1 mark]

6. What is the name of the error when a chain of calls uses up all the memory set aside for the call stack? [1 mark]

The task: watch the call stack

The starter drives the robot through `lap(10)`, which calls `corner` twice, and each `corner` calls `beep`. Make the call stack visible. - Keep a list called `stack`. At the **start** of every subroutine, push a frame onto it: a dictionary with the keys `"name"`, `"params"` (a dictionary of parameter name to value) and `"return_to"` (the name of the caller, or `"main"` for the call from the main program). Then print `push <name> <parameter>=<value>, return to <caller>, depth <d>`, where `<d>` is the number of frames on `stack` after the push. - At the **end** of every subroutine, pop the top frame and print `pop <name>, depth <d>`, where `<d>` is the number of frames left. - Keep track of the deepest the stack gets, and after `lap(10)` has finished print `deepest: <n>`. The first line is `push lap size=10, return to main, depth 1` and the last before `deepest` is `pop lap, depth 0`. The robot still drives and beeps as before.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

stack = []

def beep(freq):
    tone(freq, 0.2)

def corner(cm):
    forward(50, distance=cm)
    beep(880)

def lap(size):
    corner(size)
    corner(size)

lap(10)
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a2-1-stack-frames-and-the-call-stack/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Draw the stack, frame by frame, at the moment the second `beep` starts. Which return address is on top?
2. Change `lap` so it calls `corner` four times. Does the deepest depth change? Explain why.
3. A subroutine can return before its last line with an early `return`. What goes wrong with your model if it does, and how would you fix it?