WORKSHEET

A1.5 Scope, lifetime and debugging in an IDE

Programming techniques and object-oriented programming · A level · OCR H446 2.2.1, AQA 7517 4.1.1.13, Eduqas A500QS 1.8 · about 20 min

Name _____ Class _____ Date _____

What this lesson is about

Local and global variables, why locals are good practice, and breakpoints, stepping, watches and tracebacks.

- Scope and lifetime
- Shadowing, and changing a global
- Why local variables are good practice
- Debugging tools

Questions

6 marks in all

1. What is the lifetime of a local variable? [1 mark]

- ☐ **A** From when its subroutine is called until the subroutine returns
- ☐ **B** The whole time the program runs
- ☐ **C** Until the variable is printed
- ☐ **D** Until the next global variable is declared

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

```
speed = 80

def slow():
    speed = 30
    return speed

print(slow(), speed)
```

3. Why is it good practice to use local variables rather than global variables in subroutines? [1 mark]

Tick every answer that is true.

- ☐ **A** The subroutine can be tested and reused on its own
- ☐ **B** A value cannot be changed unexpectedly by another part of the program
- ☐ **C** The same name can be used safely in different subroutines
- ☐ **D** Local variables can be used anywhere in the program

4. A program gives the wrong total, but only on the 40th time round a loop. Which IDE feature lets you run at full speed and then pause at the line that updates the total? [1 mark]

- ☐ **A** A breakpoint
- ☐ **B** Syntax highlighting
- ☐ **C** Auto-completion
- ☐ **D** Code folding

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

```
total = 0

def add(n):
    global total
    total = total + n

add(5)
add(7)
print(total)
```

6. An exception is not handled and Python prints a traceback. What does the traceback show? [1 mark]

- ☐ **A** The chain of subroutine calls that led to the line where the exception was raised
- ☐ **B** Every variable's value
- ☐ **C** The lines that ran successfully
- ☐ **D** A suggested fix

The task: fix the scope bug

This program should creep towards the wall in 5 cm steps while `distance()` is more than 25, then print `steps: <n>`, the number of steps taken. It stops with an `UnboundLocalError`. Run it and read the message first. Fix it **without** `global`: give `step` a parameter, `count` (the number of steps so far, an integer), and make it return `count + 1`, with the main program assigning the result back. Add a watch line: before each move, `step` prints `step <n>: <cm> cm`, where `<n>` is the number of the step about to be taken (1 for the first) and `<cm>` is the value `distance()` returns.

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

steps = 0

def step():
    forward(50, distance=5)
    steps = steps + 1

while distance() > 25:
    step()
print("steps:", steps)
```

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

Do it on the robot

www.bugbotlab.com/learn/a1-5-scope-and-the-ide/

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

Challenges

1. Fix the program the other way, with `global steps`. Then list what a reader of `step` now has to check that they did not before.
2. Make `creep` in the first cell take the speed as a parameter instead of reading the global `SPEED`. What does that make easier to test?
3. Put `print(beeps)` just before `beeps = beeps + 1` in the second cell. Why does that line fail too, when reading a global normally works?