Upper Key Stage 2 - Session 6
Flying solo with Python! Programming independently using selection, defining procedures/functions

Objectives

- Design and write programs independently in Python using selection:
  while,
  if, elif, else
- Debug their own Python programs, demonstrating an understanding of the appropriate syntax
- Use indents correctly in Python

Extension

- Defining new **procedures** in Python using **def procedurename ()**:
- and incorporating repetition within the **procedure** using for **count in range (n)**:
- Use **comments** in Python to explain how the program works

Resources

- Interactive White Board (IWB)
- Levels 98 to 105 in Rapid Router
- Resource sheets UKS2-S6-1
- UKS2 Levels Guide
- UKS2 Program Solutions Table UKS2-PST
- UKS2 Assets – Python Cards

Vocabulary

Selection
- while not v.at_destination():
- if v.is_road_forward():
- elif v.is_road_right():
- else:

Procedure or function
- Defining new procedures
- def procedurename():

Calling a new procedure
- procedurename()

Repetition
- for count in the range(n):

Let's get started

Explain that in this session, the children will be building their skills as independent Python programmers. They need to try and remember the code, but they have Python Cards to remind them and they can always go back to earlier Blockly levels to check what the code in Python looks like.

Their challenge is to use the idea of selection in programming, and to improve their de-bugging skills.

Show the class Level 98 on the IWB and read the level instructions together. *[fig S6.1]*

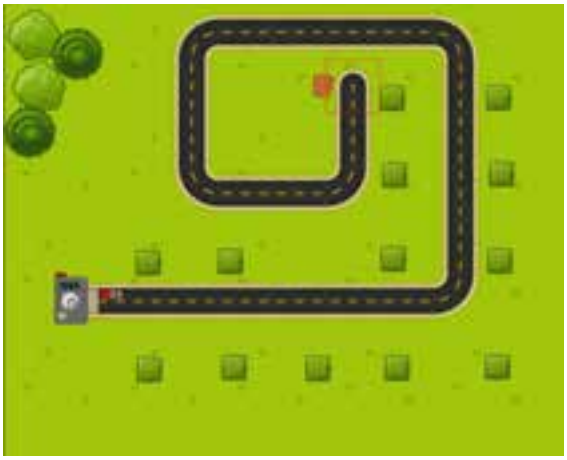

fig S6.1

Split the children up into pairs and ask them to think about everything they have learnt about programming. You may also want to suggest the most efficient way to tackle this level. Give the children some time to discuss it with their partners.

You are likely to get a range of responses, which will hopefully include the **'repeat until at destination'** loop which they generated in Blockly. The equivalent in Python is the **'while'** loop, which they looked at in Session 4, see Levels 85 and 86. If needed, go back over these levels.

```
import van

v = van.Van()
while not v.at_destination()
  if v.is_road_forward()
    v.move_forwards()
  else
    v.turn_left()
```

fig S6.2

How can we write a 'while not at destination' loop in Python?

Try the code in Python together, but make some deliberate mistakes to enable the class to debug together. For example, leave out the colons at the end of the **while** and **if** statements. *[fig S6.2]*

They will see the error message for the problem per line of code, e.g.:

ParseError: bad input on line 5

Why do we need a colon at the end of the while, if, elif and else lines of code?

Because there is an action that Python must execute if that condition is met.

Then see what happens if you leave out the indents. *[fig S6.3]*

```
import van

v = van.Van()
while not v.at_destination():
if v.is_road_forward():
v.move_forwards()
else :
v.turn_left()
```

fig S6.3

Why do you think the indents are important in the program?

They are part of the structure of the while loop, without them Python can't follow the code.

Guide them through it so you end up with a correct solution, emphasising that debugging is a very important skill. *[fig S6.4]*

```
import van

v = van.Van()
while not v.at_destination():
  if v.is_road_forward():
    v.move_forwards()
  else :
    v.turn_left()
```

fig S6.4

Break up the group into pairs and give the children a Level 98 sheet from the Levels Guide and one Resource Sheet UKS2-S6-1. Ask each pair to discuss and write code to solve this level.

If some children find it too great a challenge to write code in Python without support, give them Resource Sheet UKS2-S6-2, which shows the skeleton of the code for help.

Main activity

Ask the children to attempt Levels 98 to 100, using the Resource Sheets UKS2-S6-1 and UKS2-S6-2 to help if needed.

Share and review

Choose a pair of children to feedback, explaining what they have learnt about the way you must write the code in Python. Ask a child to record the main points on a flipchart (e.g. underscore, brackets, colon, indents..)

Extension - Define your own procedures

Procedures are also known as **functions** in Python.

The challenge in Levels 101 to 104 is to spot a section of code which is used more than once.

You can then define a new procedure to represent this code.

Show Level 101 on the IWB, to those in the class ready to take this on. *[fig S6.5]*

Can you spot a short piece of code which is used in several places?

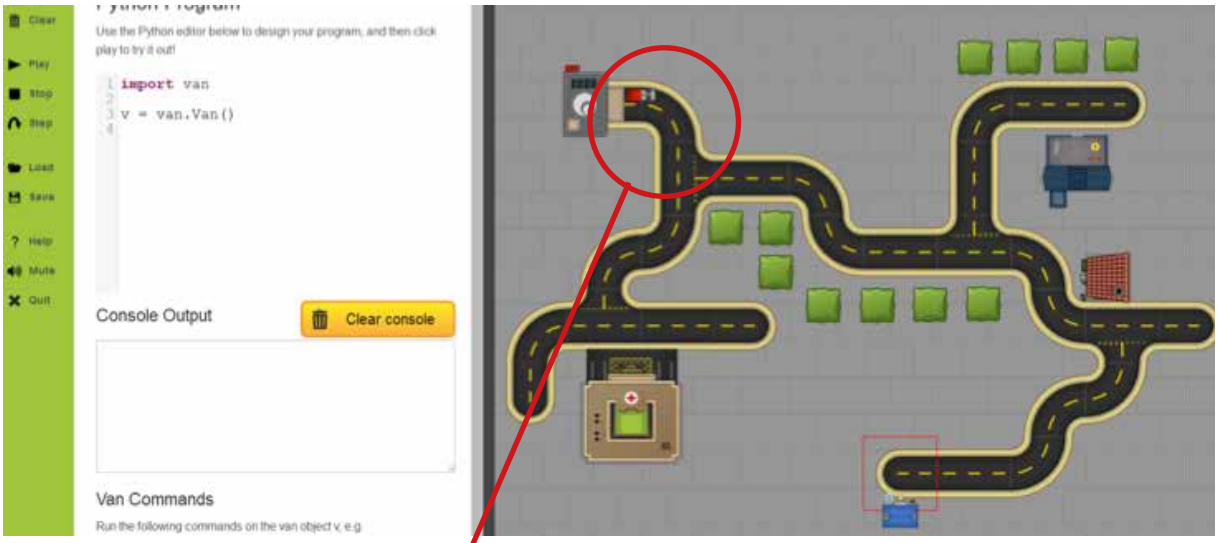

fig S6.5

Identify the right-left wiggle on the road.

We can define this as a **procedure**. But we need to give the **procedure** a name – for example **right_left**:

```
import van

v = van.Van()

def right_left():
  v.turn_right()
  v.turn_left()

```

Can you see how the procedure name you chose has to be followed by a bracket and colon? What do you notice about the instructions which form the procedure? (they are indented)

We can write code using this new procedure. You may need to use a repeat loop … can you see where?

```
right_left()
v.move_forwards()
right_left()
for count in range(2):
  v.move_forwards()
for count in range(2):
  right_left()
  v.turn_right()
v.move_forwards()
```

Give the children a chance to create their own **procedures** in Levels 101 to 105. You may set some of this as homework.

Extension activity

You might want to teach the children how to add a **comment** to their program. A **comment** is explanatory text to make your code clear to other programmers. It is common practice that programmers, other than the person who wrote it, may need to go back and later edit the code.

To create a comment, you must put a hash symbol **#** in front of your text so that Python knows it's just information for the reader and not code for Python to action.

You can use a **comment** as an assessment tool. Ask the children to add a **comment** to explain why they decided to define a particular **procedure**.

```
import van

v = van.Van()

# right_left is my procedure for the wiggle I can see in the maze

def right_left ():
  v.turn_right
  v.turn_left

```