Introduction to Programming for Physicists

Dr. Charanjit Kaur

charanjit.kaur@manchester.ac.uk

University of Manchester

PHYS20161 Pre-lecture 3
Semester 1, 2023-24

The material is adapted from previous year's course

Learning Objectives

After covering week 3 material, you will get to know

- What is recursion in programming.
- Iterative Processes
  - While Loops
  - For Loops
- Formatted Output
- Debugging
- Coding Style

Plan

- Part 1: Recursion [Video 1, `factorial_recursion.py`]
- Part 2: `while` loop [Video 2, `while_count_5.py`]
- Part 3: `for` loop [Video 3, `for_count_5.py`]

`factorial_examples.py`

- Part 4: Formatting output [Video 4]
- Part 5: Debugging [Video 5]
- Part 6: Style [Video 6]

Part 1

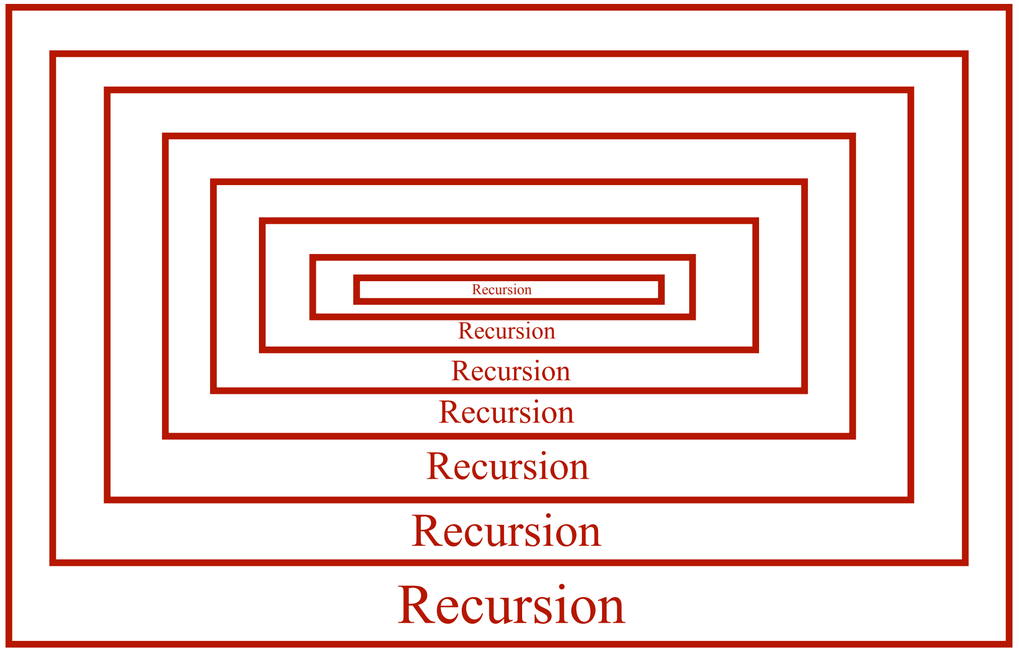
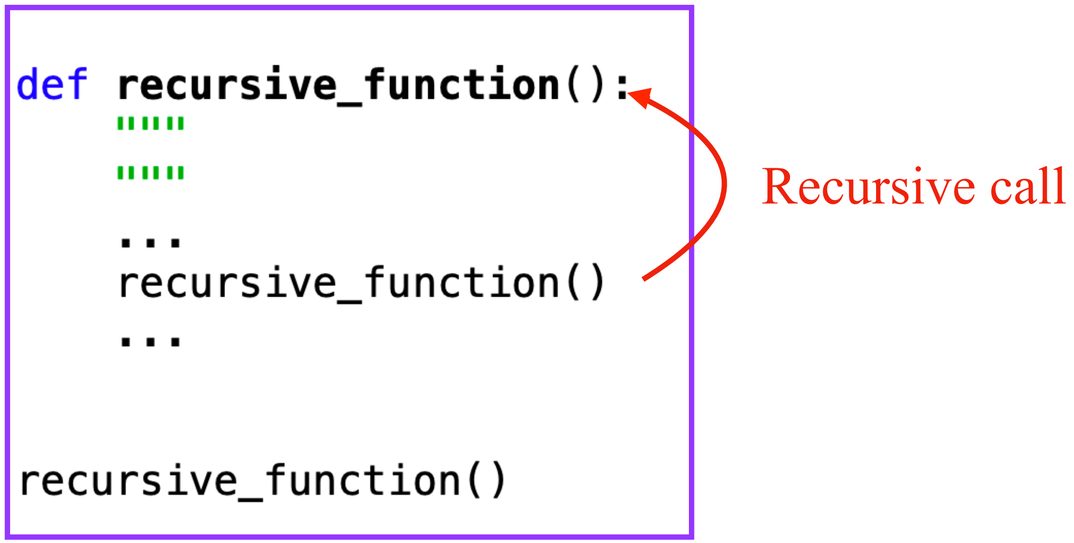


Figure adapted from https://blog.devgenius.io/

`factorial_recursion.py`

Recursion

- The process of defining something in terms of itself.
- A function calling itself is called recursion and that function is called recursive function.

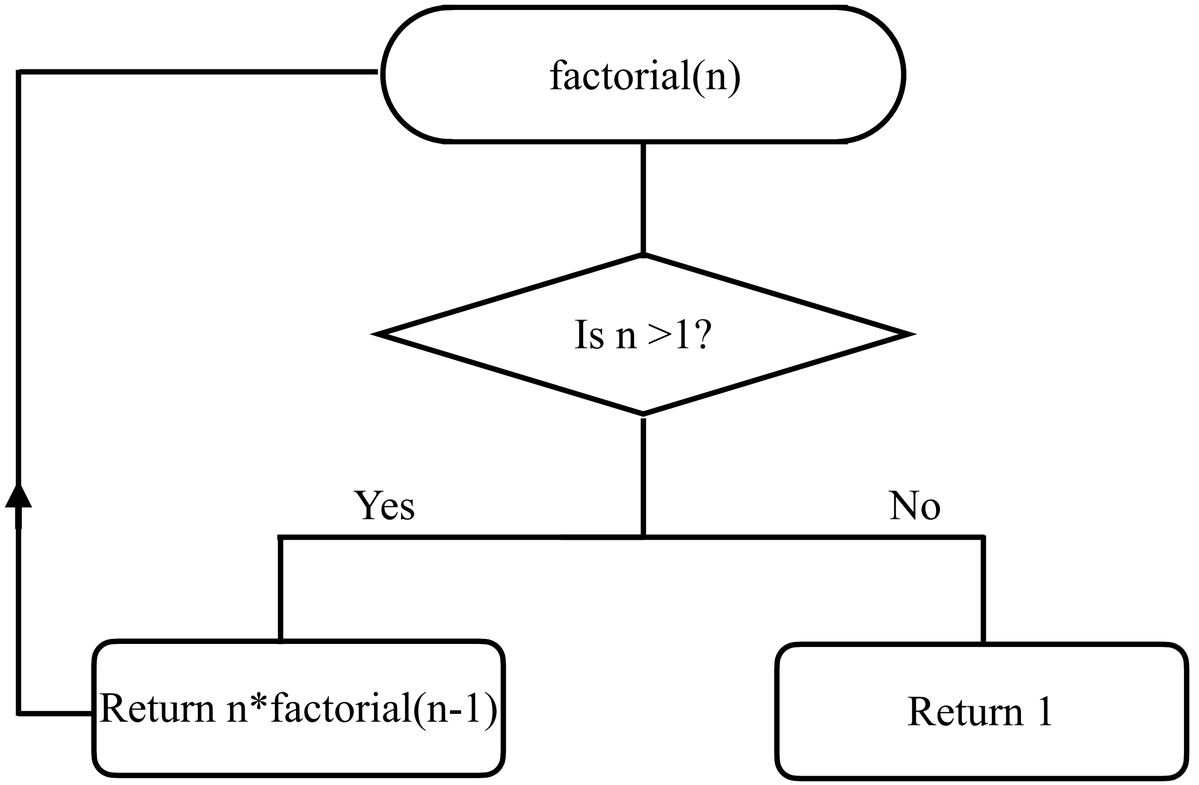


- Example: calculating factorial of an integer.

Recursive Function: `factorial`

```
11
12  def factorial(number):
13      """
14      Returns the factorial of a number using recursion
15      Parameters
16      ----------
17      number : int
18
19      Returns : int
20      """
21      if number > 1:
22
23          return number * factorial(number-1)
24
25      else:
26
27          return 1
28
```

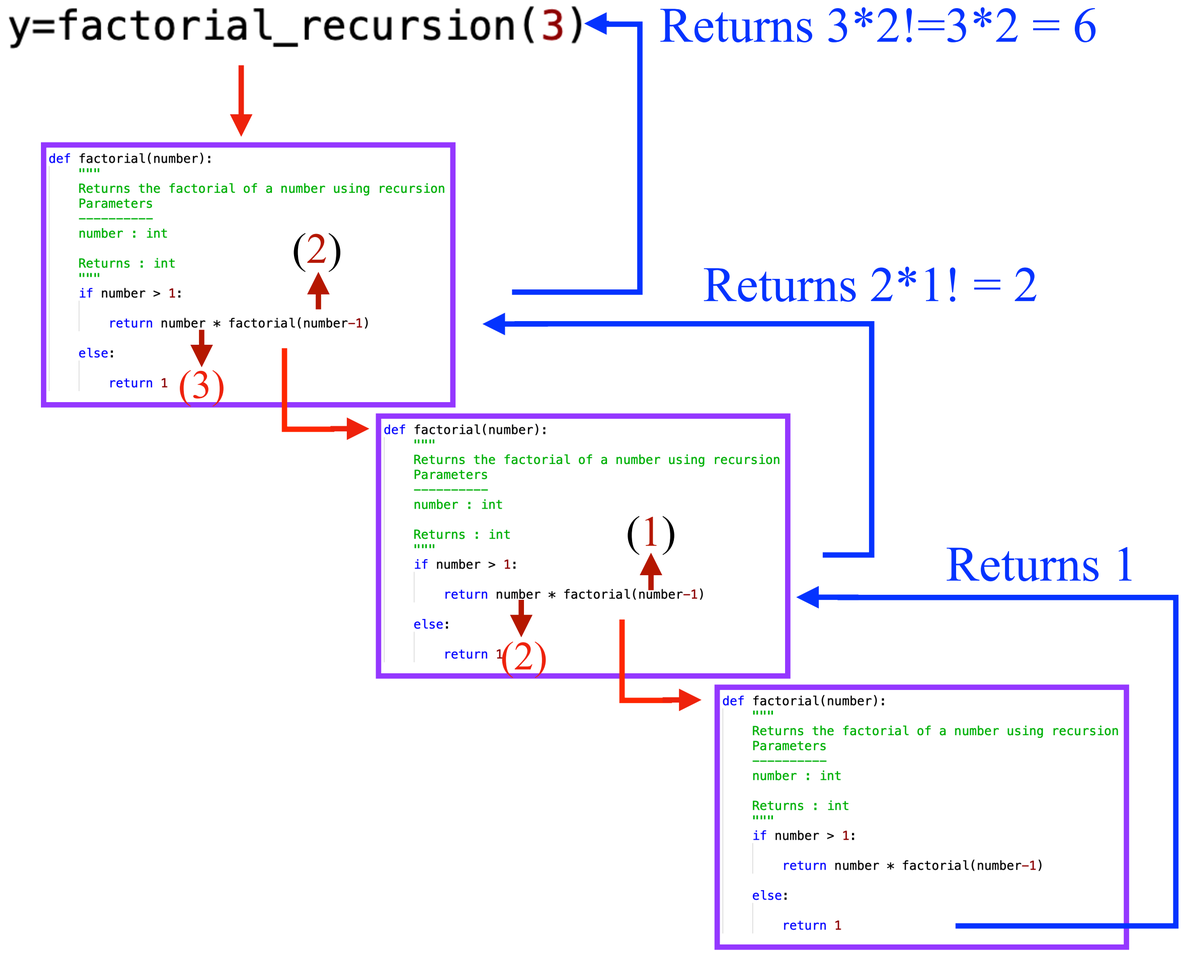
PL3: selection of factorial_recursion.py



Recursion : Illustration

```
12  def factorial(number):
13      """
14      Returns the factorial of a number using recursion
15      Parameters
16      ----------
17      number : int
18
19      Returns : int
20      """
21      if number > 1:
22
23          return number * factorial(number-1)
24
25      else:
26
27          return 1
28
```

PL3: selection of factorial_recursion.py

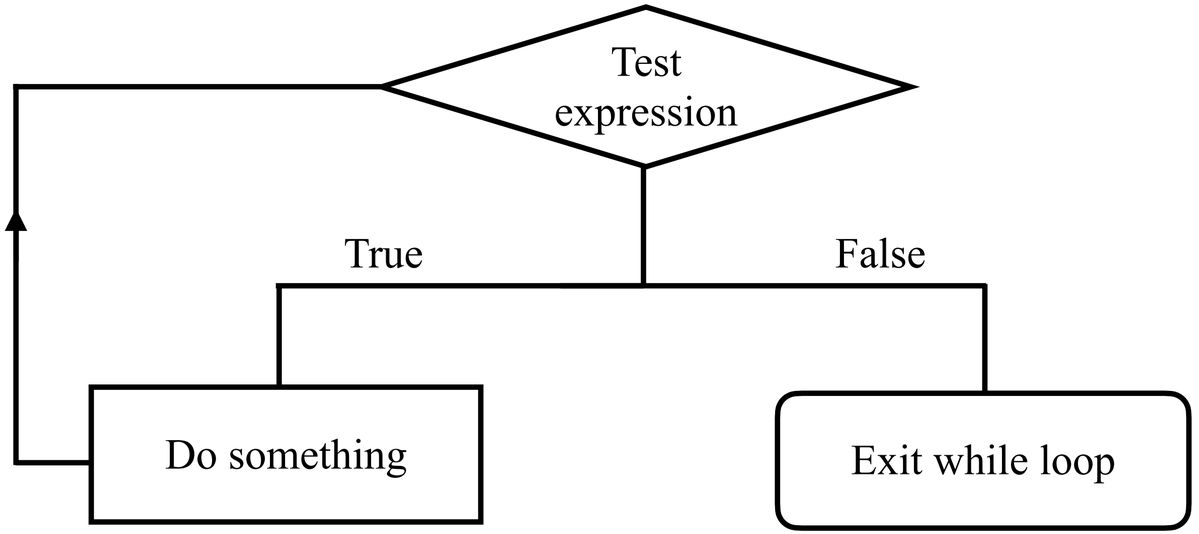


Part 2
`while` loop

`while_count_5.py`

while Loop

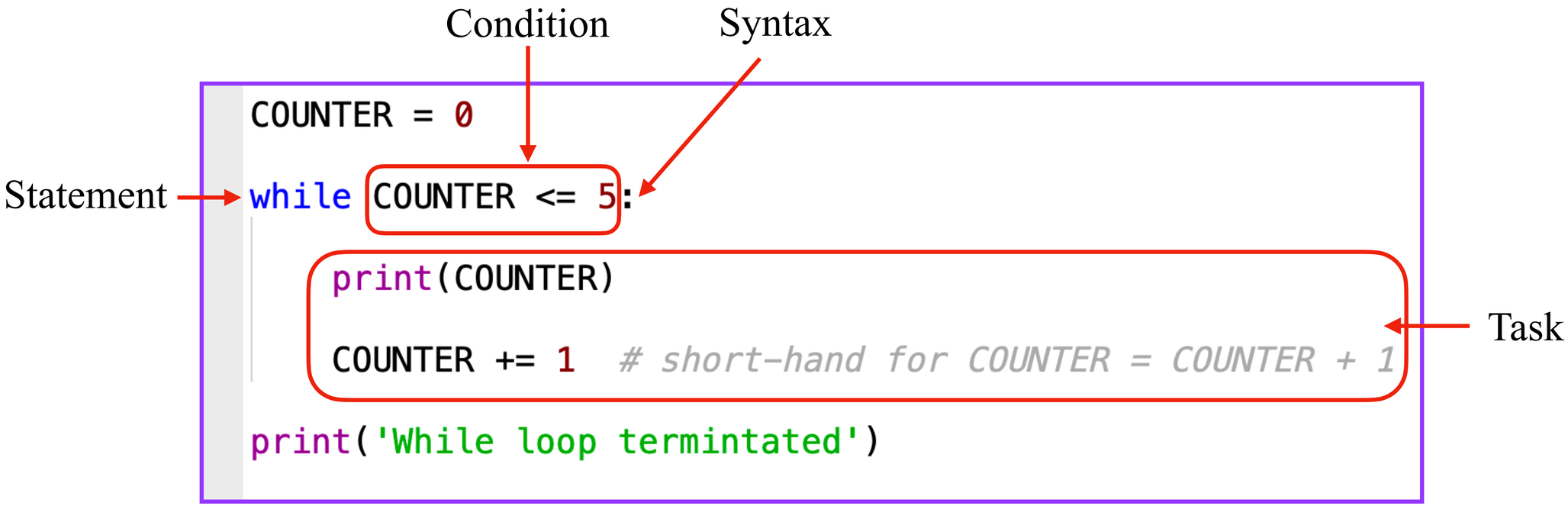
- Perform task/operation repeatedly until condition is no longer true.




```
COUNTER = 0

while COUNTER <= 5:

    print(COUNTER)

    COUNTER += 1  # short-hand for COUNTER = COUNTER + 1

print('While loop termintated')
```

PL3: selection of while_count_5.py

while loop



PL3: selection of while_count_5.py

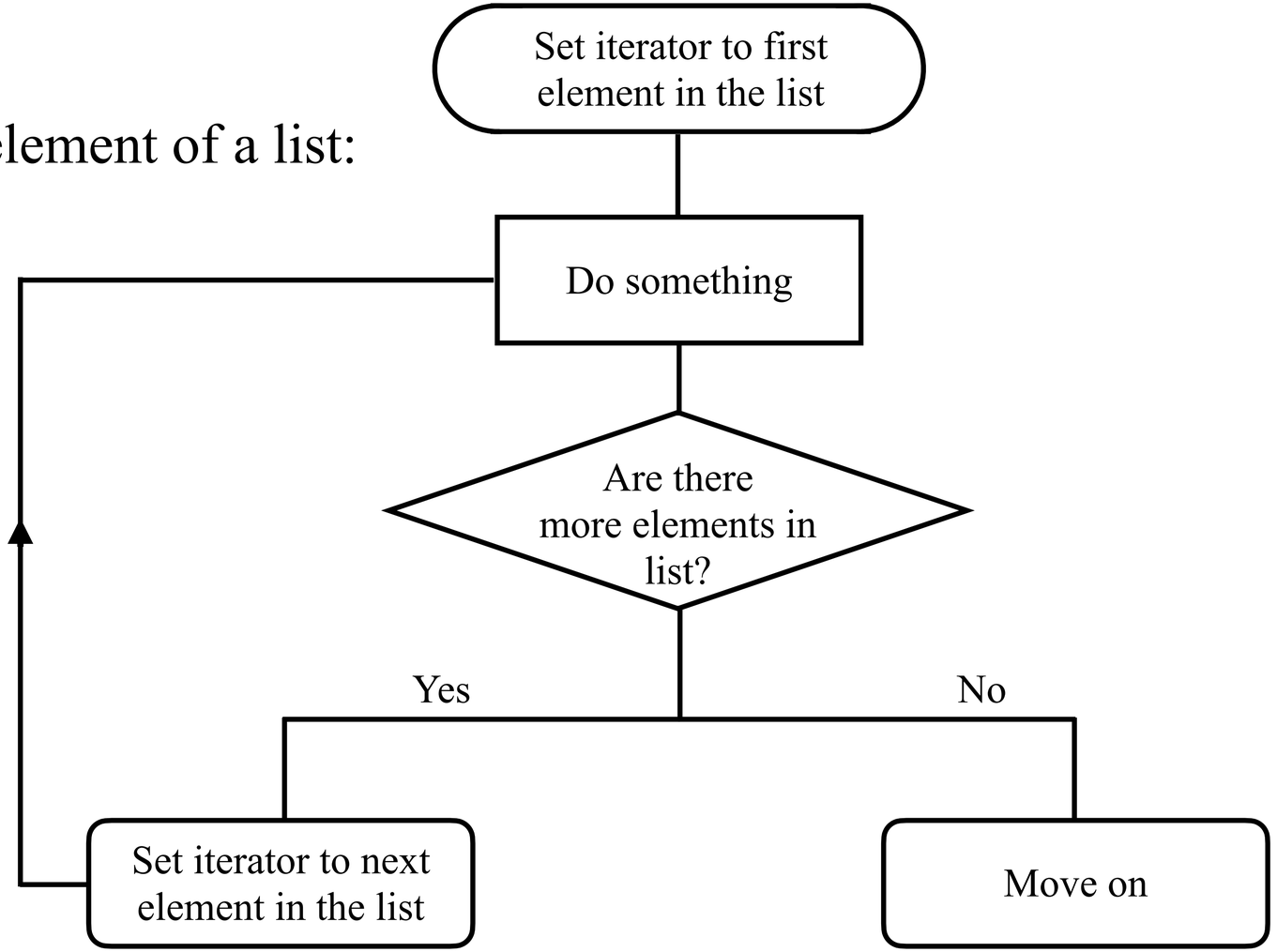
Part 3
`for` loop
`for_count_5.py`

for loop

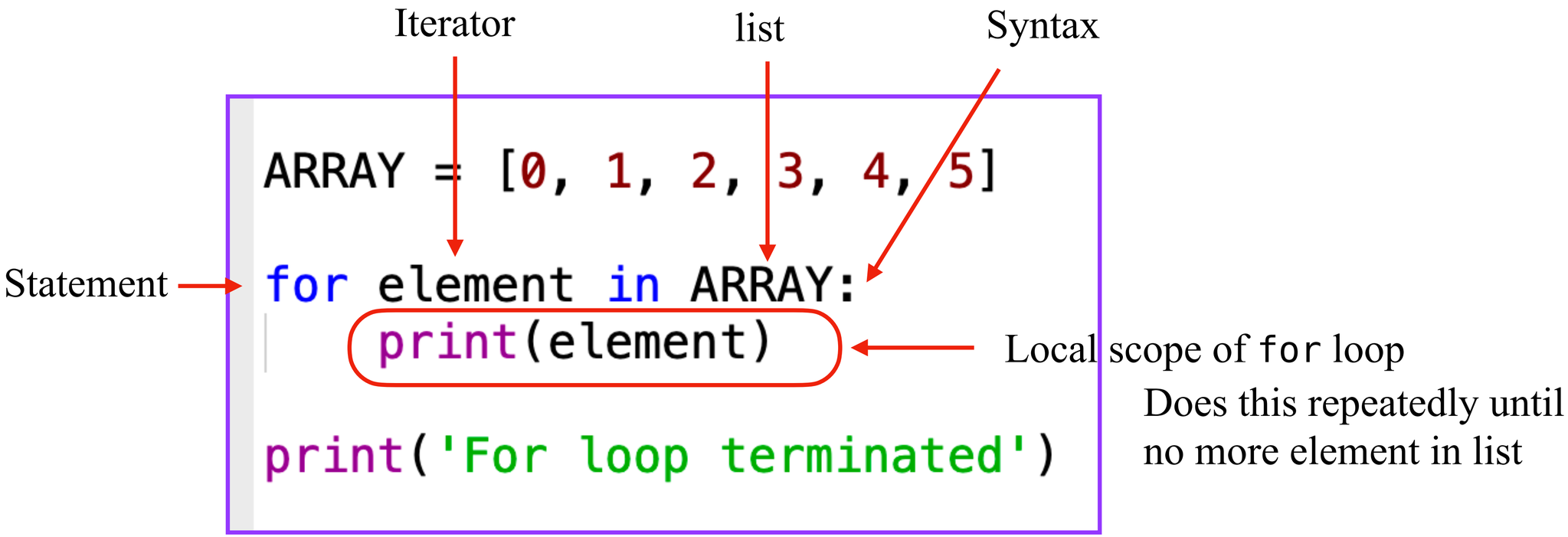
- They loop through a collection of things.
- Example: perform an operation for each element of a list:

```
ARRAY = [0, 1, 2, 3, 4, 5]

for element in ARRAY:
    print(element)

print('For loop terminated')
```

PL3: selection of for_count_5.py



for loop



PL3: selection of for_count_5.py

range(start,stop,step)

Creates a sequence based on integer arguments:

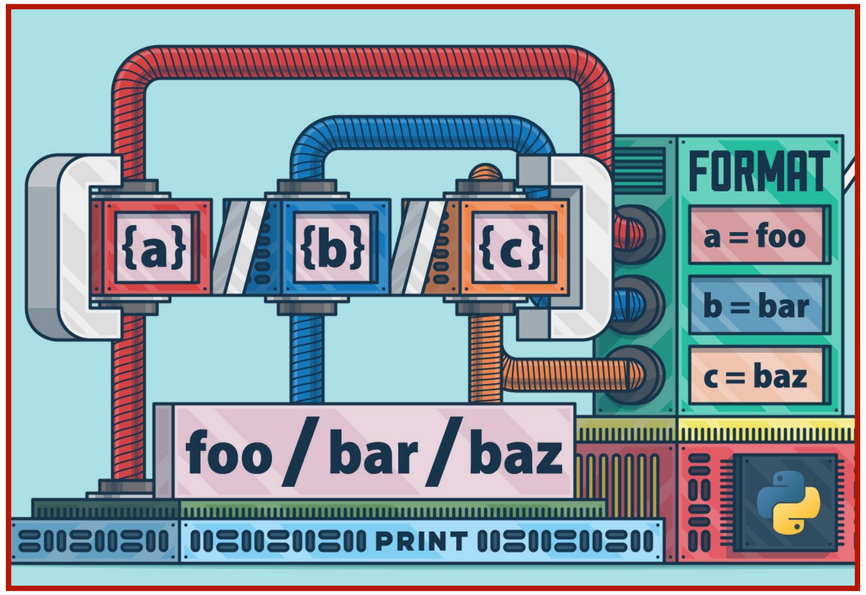
- Start: optional, integer defining start of array, default is 0.
- Stop: required, integer specifying at which position to end.
- Step: optional, integer defining increment between elements, default is 1.

```
In [1]: for i in range(4):
   ...:     print(i)
   ...:
0
1
2
3
```

```
In [2]: for i in range(1,4):
   ...:     print(i)
   ...:
1
2
3
```

```
In [3]: for i in range(1,7,2):
   ...:     print(i)
   ...:
1
3
5
```

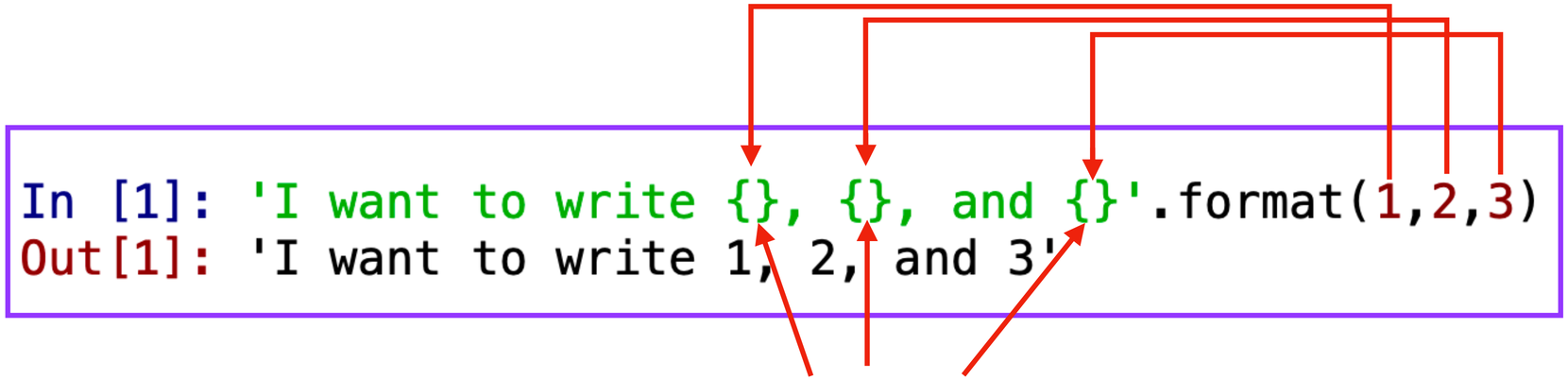
Part 4
Formatted Output



Source : Real Python (adapted)

.format()

- A string format method to insert objects into string:



- We can also number or name the replacement fields:

```
In [2]: 'I want to write {0}, {1}, and {2}'.format(1,2,3)
Out[2]: 'I want to write 1, 2, and 3'
```

```
In [3]: 'I want to write {num3}, {num2}, and {num1}'.format(num1=1, num2=2, num3=3)
Out[3]: 'I want to write 3, 2, and 1'
```

Examples .format()

- A string format method to insert objects into string:

```
In [1]: '{} plus {} = {}'.format(2,2,4)
Out[1]: '2 plus 2 = 4'
```

- We can also have numbers in the replacement fields:

```
In [2]: '{} plus {} = {}'.format('Two',2,'four')
Out[2]: 'Two plus 2 = four'
```

```
In [3]: '{1} plus {0} = {2}'.format('Two',2,'four')
Out[3]: '2 plus Two = four'
```

```
In [4]: '{0} plus {0} = {1}'.format(2,'four')
Out[4]: '2 plus 2 = four'
```

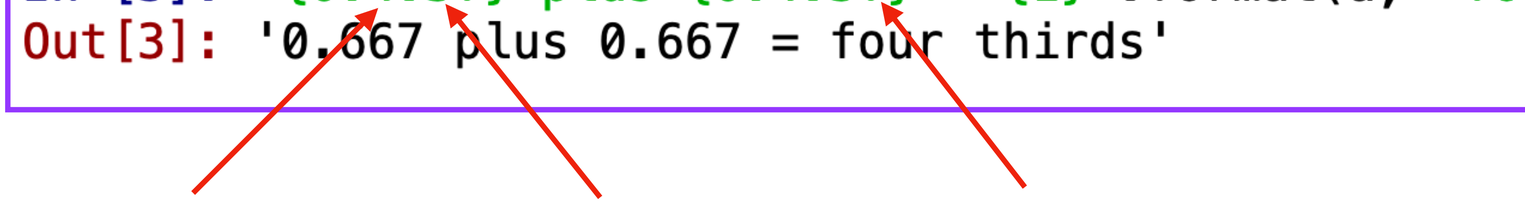
Formatting Numbers

- We want to avoid:

```
In [1]: a = 2./3.

In [2]: '{0} plus {0} = {1}'.format(a, 'four thirds')
Out[2]: '0.6666666666666666 plus 0.6666666666666666 = four thirds'
```

- Specify precision for floats:

```
In [3]: '{0:4.3f} plus {0:4.3f} = {1}'.format(a, 'four thirds')
Out[3]: '0.667 plus 0.667 = four thirds'
```



Formatting Numbers Cont.

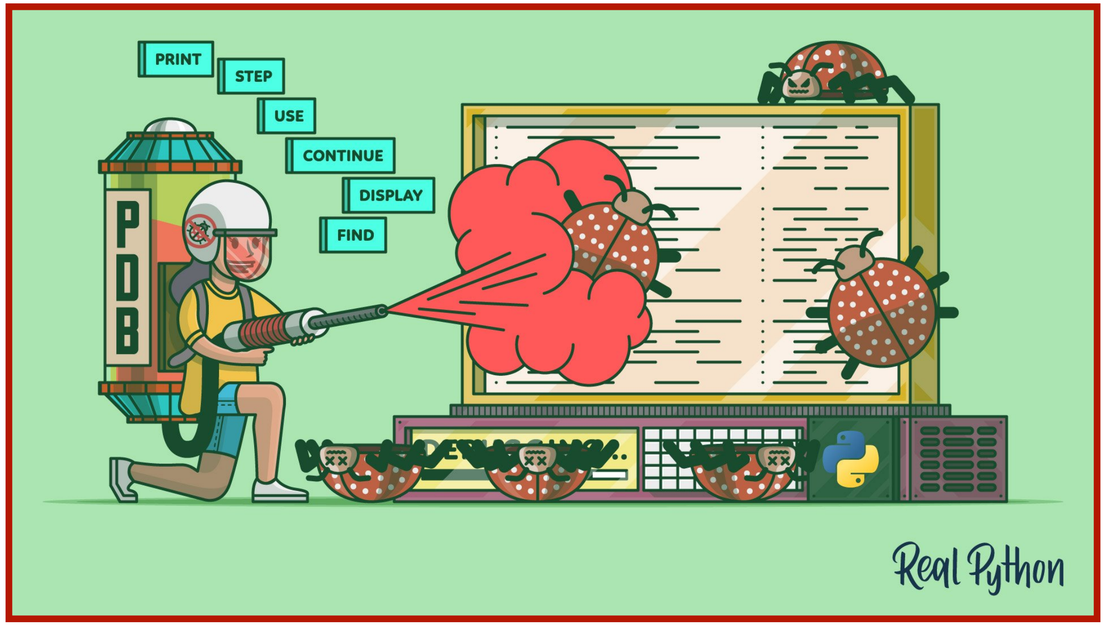
- We can choose different styles:

```
In [4]: '{0:4.3E} plus {0:4.3E} = {1}'.format(a,'four thirds')
Out[4]: '6.667E-01 plus 6.667E-01 = four thirds'
```

- We can have rounding errors:

```
In [5]: a = 0.00000000678

In [6]: 'a = {0:4.3f}'.format(a)
Out[6]: 'a = 0.000'

In [7]: 'a = {0:12.11f}'.format(a)
Out[7]: 'a = 0.00000000678'

In [8]: 'a = {0:3.2e}'.format(a)
Out[8]: 'a = 6.78e-09'
```

Part 5



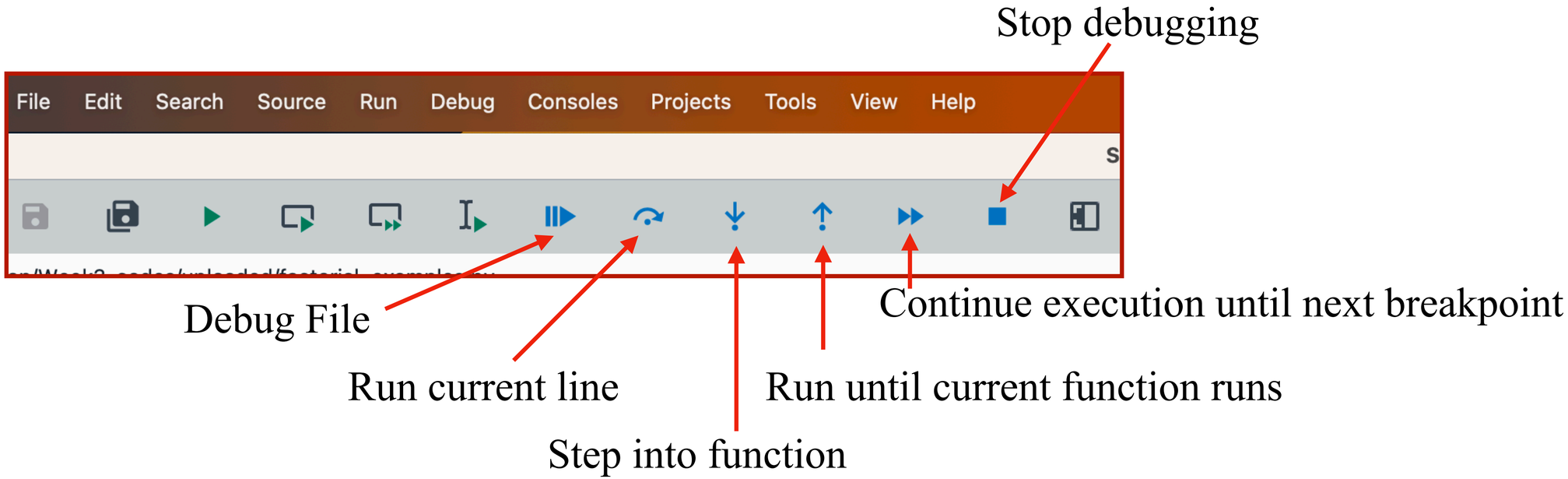
Debugging

Debugging and Testing

- It is impossible to write a programme that works perfectly in one go.
- Often it will not work correctly and we have to find the mistakes. This takes some practice (hence some of the BB quiz questions).
- We can employ some useful approaches to help us to ensure different aspects are working correctly:
  - Print statements
  - Debugging mode

Debugging in Spyder

- PDB: Python DeBugger



Part 6
Coding Style

Style

- Good coding style is subjective. Though there are general aspects that are agreed upon.
- Good code is
  - self-explanatory; does not need to reply on comments.
  - straightforward; is not over complicated.
  - organised; has small modular blocks that can be reused.
  - can be read and understood in a short amount of time with minimal thinking.

We have a style guide describing what we expect from your code with examples.

Different Styles

```
a = [4, 2, 5, 8, 7, 1, 3, 9, 7]

a2 = [] # a times 2

for x in a:
    a2.append(x * 2)

print(a2)
```

PL3: example code A

```
list1 = [4, 2, 5, 8, 7, 1, 3, 9, 7]

list1_times_2 = []

for element in list1:
    list1_times_2.append(element * 2)

print(list1_times_2)
```

PL3: example code B

Code B is better written because it has clear variable names.

Different Styles

```
import numpy as np

angles = [0.145, 6.2, 1.23, 0.87, 14.68, 4.7, 4.385]
tangents = []

for angle in angles:

    tan_temp = np.tan(angle)
    tangents.append(tan_temp)

    print('tan(angle) = {0:4.2f}'.format(tan_temp))
```

PL3: example code A

```
import numpy as np

angles = [0.145, 6.2, 1.23, 0.87, 14.68, 4.7, 4.385]
tangents = []

for index in range(len(angles)):

    sin_temp = np.sin(angles[index])
    cos_temp = np.cos(angles[index])
    tan_temp = sin_temp / cos_temp
    tangents.append(tan_temp)

for tangent in tangents:

    print('tan(angle) = {0:4.2f}'.format(tangent))
```

PL3: example code B

Code A is better written because it performs minimal operations for the same task.

Summary

- Recursion is a process of defining something in terms of itself.
- We can iterate processes using for and while loops.
- Formatted output is more readable.
- We can use debugging mode or useful print statements to check if code is functioning as desired.
- Good style makes code easier to read and understand.
- We can use static code analysis to check code against PEP8 standards.

Next week, we will cover code validation and start using NumPy (and NumPy arrays).