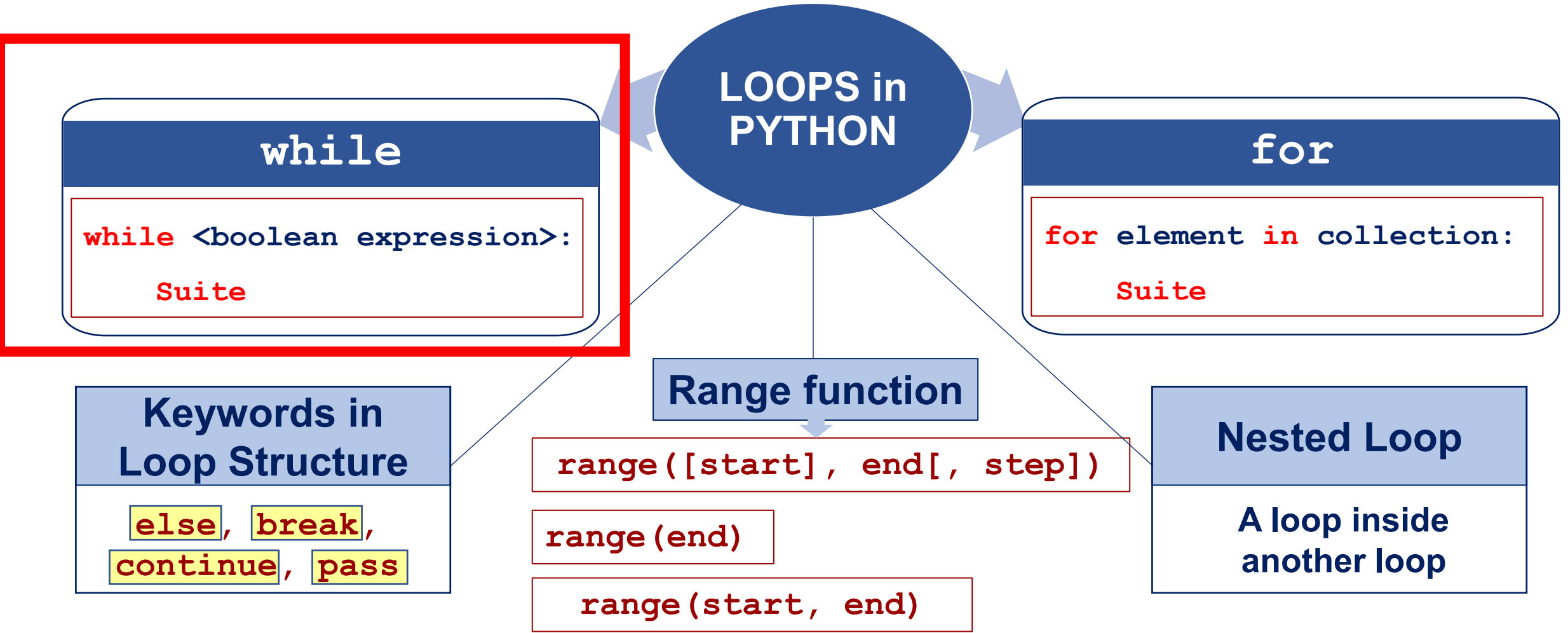




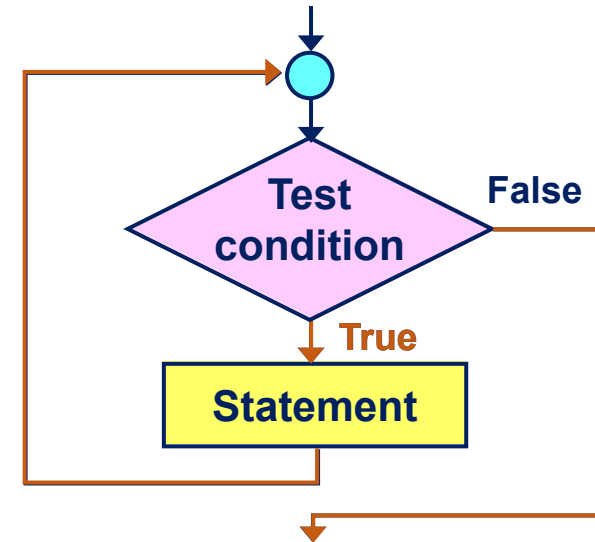
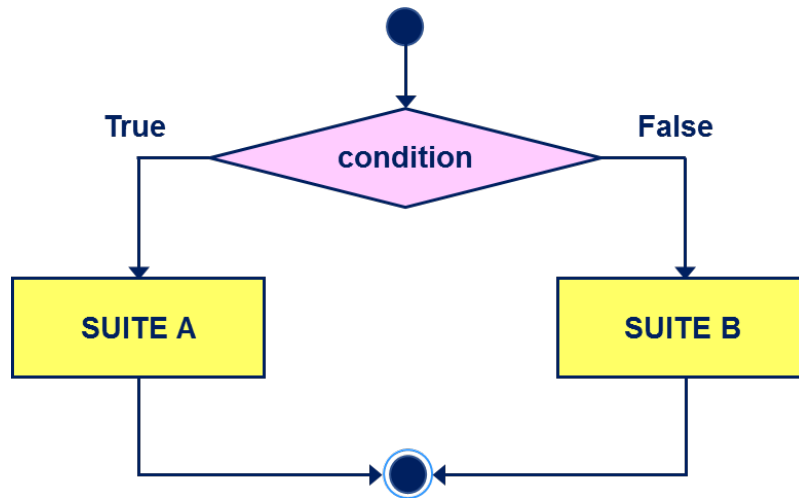
Course Review (looping)

Summary



Loops in Python: **while**

- The **while** statement allows repetition a suite of Python codes as long as a condition (Boolean expression) is True.
- It is structurally similar to an **if** statement but repeats the block until the condition becomes False.



- When the condition becomes False, repetition ends and control moves on to the code following the repetition.

Loops in Python: **while** - Syntax

The whole
while structure

```
while <boolean expression>:
    Suite (one or more indented statements)
```

indentation

It **must** use a **colon**
followed by a proper
indentation

General Execution of a Loop

Four Steps

1. Initialize

Loop control variable

2. Test

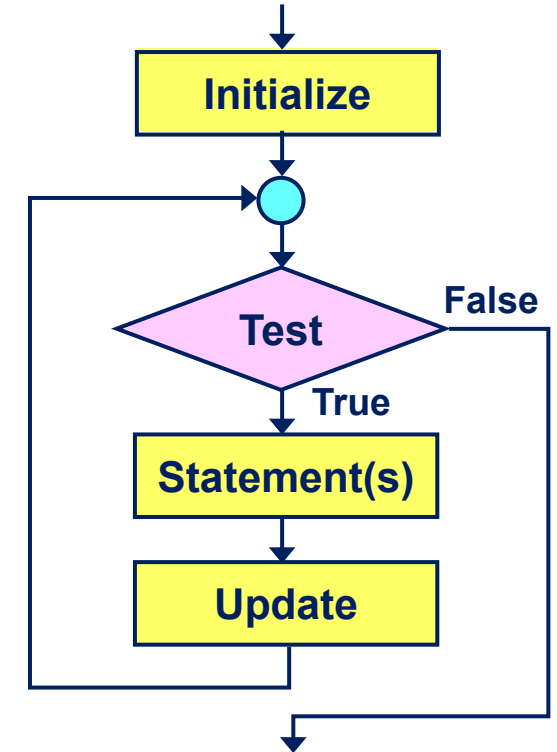
Continue the loop or not?

3. Loop body

Main computation being repeated

4. Update

Modify the value of the loop control variable so that next time when we test, we may exit the loop



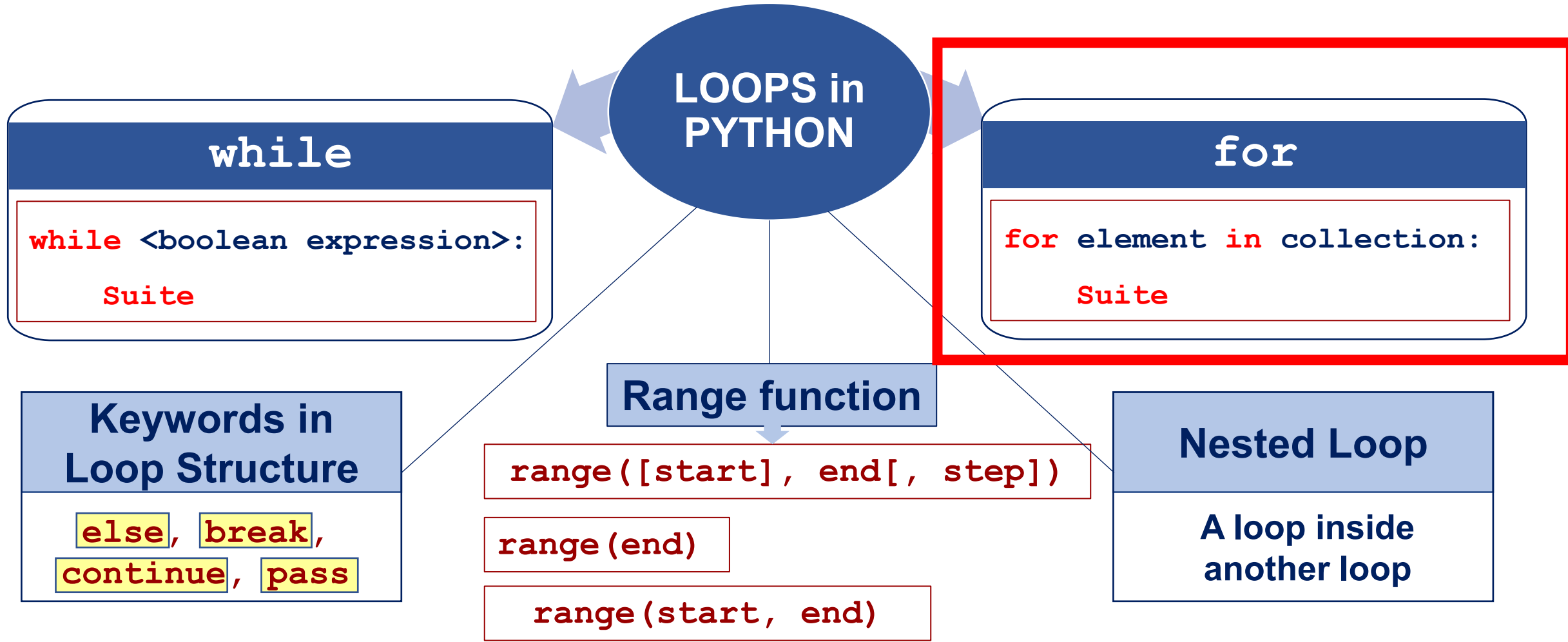
- Sometimes a loop may not have all of them. E.g., *Infinite loop* (Test condition is always true).
- A one-time execution of a loop body is referred to as an **iteration** of the loop.

What is the output of the following code?

```
m = 0
while m < 4:
    print("begin=", m, end = ", ")
    m += 2
    print("end=", m, end = ", ")
```

- A. begin= 0, end= 1, begin= 2, end= 3,
- B. begin= 0, end= 2, begin= 1, end= 3,
- ✓ C. begin= 0, end= 2, begin= 2, end= 4,
- D. begin= 1, end= 3, begin= 2, end= 4,

Summary



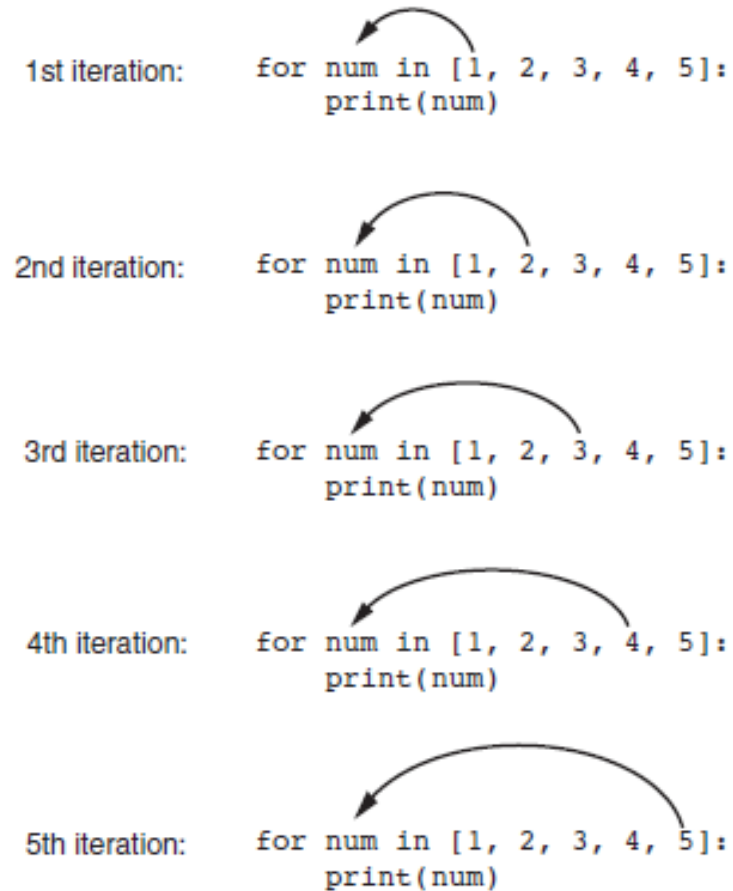
The `for` Loop: a Count-Controlled Loop

Here `<variable>` is a variable that is used for iterating over a `<sequence>`. On every iteration it takes the next value from `<sequence>` until the end of sequence is reached.

- Count-Controlled loop: iterates a specific number of times
 - Use a `for` statement to write count-controlled loop
 - Designed to work with sequence of data items
 - Iterates once for each item in the sequence
 - General format:

```
for variable in [val1, val2, etc]:  
    statements
```


Figure 4-4 The for loop



The variable `iterating_var` is assigned a different element during each pass of the for loop. Eventually, `iterating_var` will be assigned to each element in the sequence.

Loops in Python: **for** - Syntax

```
for iterating_var in <sequence>:  
    Suite (one or more indented statements)
```

indentation

It must use a **colon** followed by proper **indentation**.

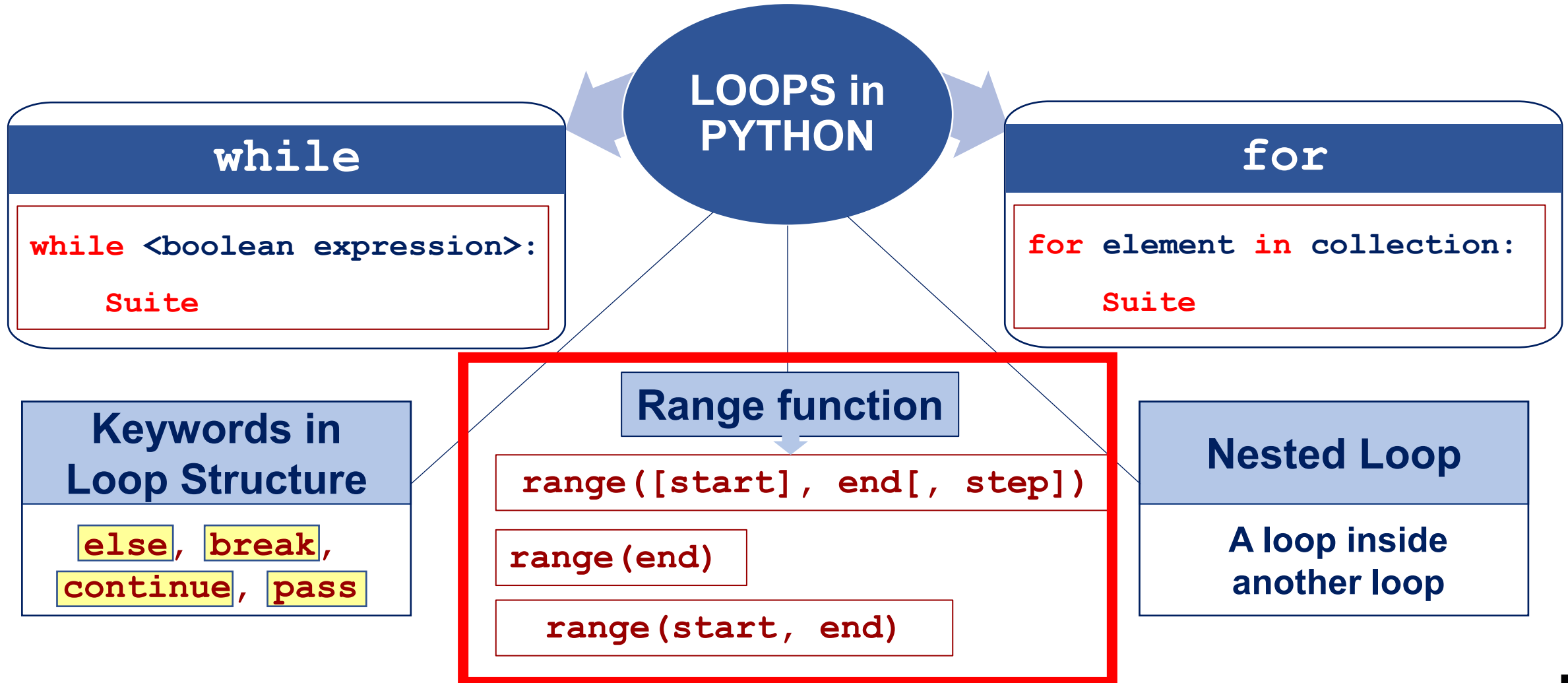
- Has a header and an associated suite.
- Keywords: **for** and **in**.
- The keyword **in** precedes the sequence.
- The variable `iterating_var` is a variable associated with the **for** loop that is assigned the value of an element in the sequence.
 - The variable `iterating_var` is assigned a different element during each pass of the **for** loop.
 - Eventually, `iterating_var` will be assigned to each element in the sequence.

What is the output of the following code?

```
for i in ["Hello", "Hi"]:  
    print("Python", end = "*")
```

- A. Hello*Hi*
- B. Hello*
- ✓ C. Python*Python*
- D. Python*
- E. HelloHiPython*
- F. None of above

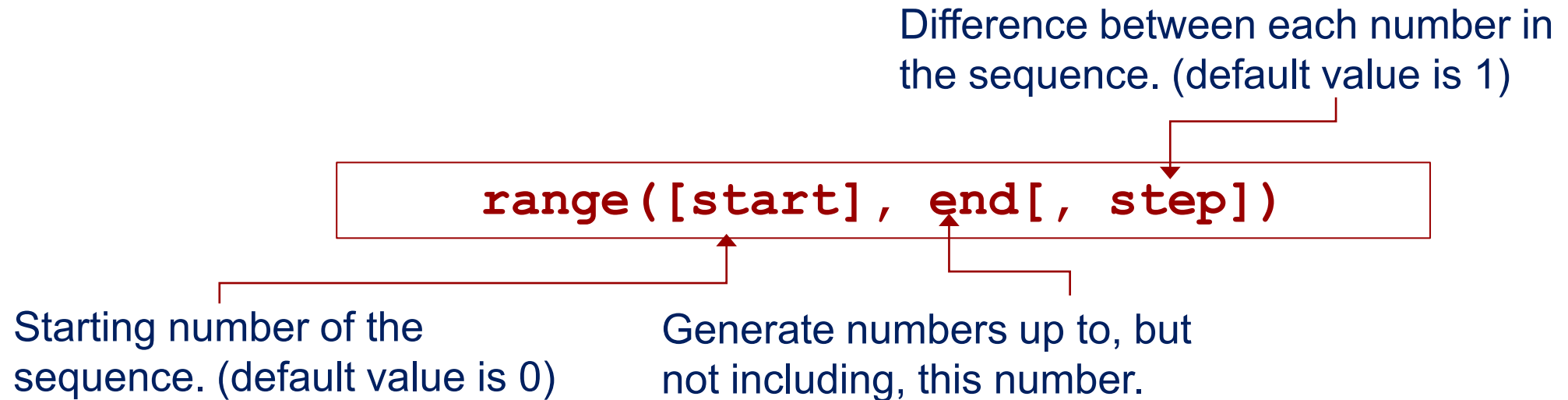
Summary



Python: The Range Function

range()

- It is a useful built-in function in Python.
- It generates a list of **integers** from **start** up to **end** (but excluding end) with step-size **step**.



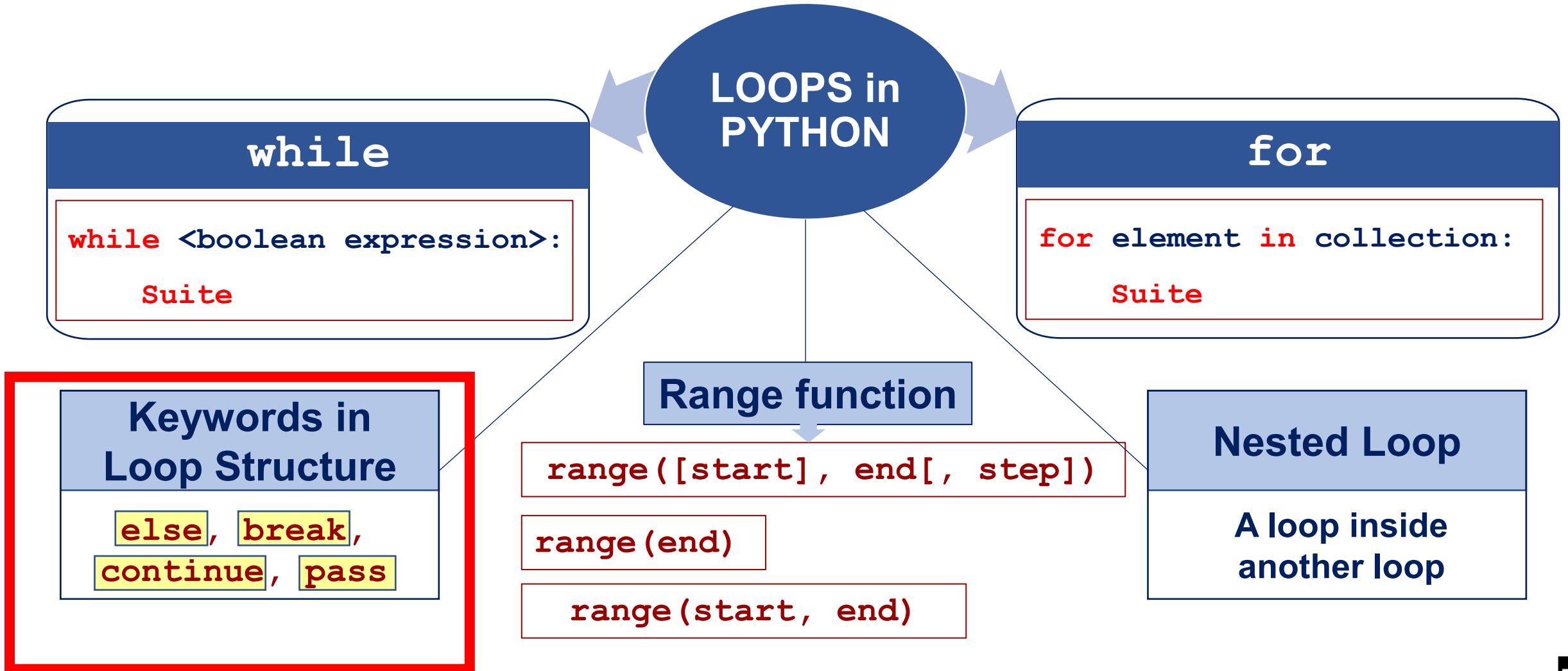
Note: All parameters must be integers, positive or negative.

What is the output of the following for loop and range() function?

```
for num in range(-2, -8, -2):  
    print(num, end=", ")
```

- A. -2, -3, -4, -5, -6, -7
- B. -8, -6, -4,
- C. -2, -1, 0
- ✓ D. -2, -4, -6,
- E. -2, -4, -6, -8,

Summary



Loops in Python: **while-else** (Syntax)

The whole
while structure

```
while boolean expression:  
    handle_true()  
else:  
    handle_false() ←  
rest of the programme
```

Condition is false now,
handle and go on with
the rest of the program

- **while** loop, can have an associated **else** statement
- **else** statement is executed when the loop finishes under **normal** conditions
 - the last thing the loop does as it exits

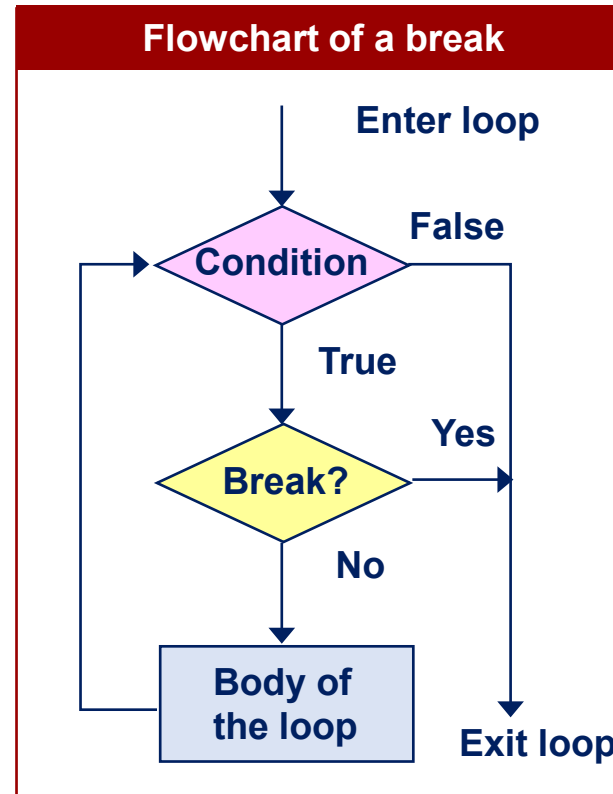
What is the output of the following?

```
i = 7
while i < 9:
    print(i, end=", ")
    i += 1
else:
    print(0)
```

- A. 7,8,0,
- ✓ B. 7,8,0
- C. 7,8,
- D. 7,8,9

Break Statement

- The **break** statement can be used to **immediately** exit the execution of the **current** loop and skip past all the remaining parts of the loop suite.
- The **break** statement is useful for stopping computation when the “answer” has been found or when continuing the computation is otherwise useless.



Working of break in a while loop

```
while test expression:  
    body of while  
    if condition  
        break  
    body of while  
→ statements
```

What is the output of the following code?

```
value = 1
print("before ", value, end=" ", )

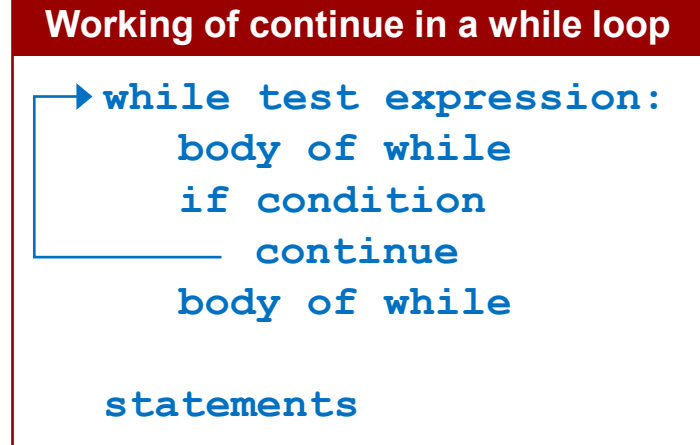
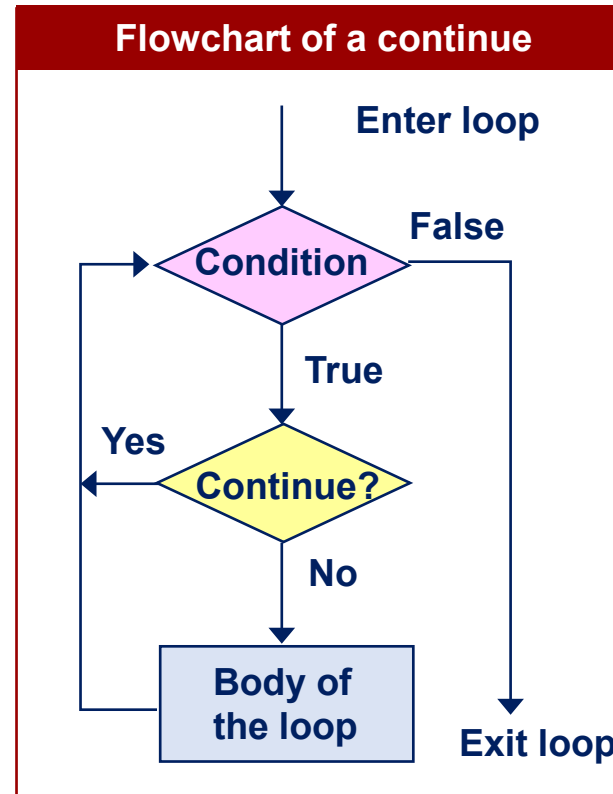
while value <=3:
    value +=1
    if value == 2:
        break
    print("while", value, end =" ", )
else:
    print("else", value, end =" ", )

print("after ", value, end =" ", )
```

- A. before 1, while 2, after 2,
- ✓ B. before 1, after 2,
- C. before 1, else 3, after 4,
- D. before 1, while 3, while 4, else 4, after 4,

Continue Statement

- Skip some portion of the **while** suite we are executing and have control flow back to the beginning of the **while** loop.
- Exit early from this iteration of the loop (not the loop itself), and keep executing the **while** loop.
- The **continue** statement continues with the next iteration of the loop.



What is the output of the following code?

```
value = 1
print("before ", value, end=" ", " ")

while value <=3:
    value +=1
    if value == 2:
        continue
    print("while", value, end =" ", " ")
else:
    print("else", value, end =" ", " ")

print("after ", value, end =" ", " ")
```

- A. before 1, while 2, after 2,
- B. before 1, after 2,
- C. before 1, else 3, after 4,
- ✓ D. before 1, while 3, while 4, else 4, after 4,

for-else-break-continue

```
for target in object:
```

```
    # statement suite1
```

```
    if boolean expression1:
```

```
        break # Exit loop now; skip else
```

```
    if boolean expression2:
```

```
        continue # Go to top of loop now
```

```
else:
```

```
    # statement suite2
```

What is the output of the following code?

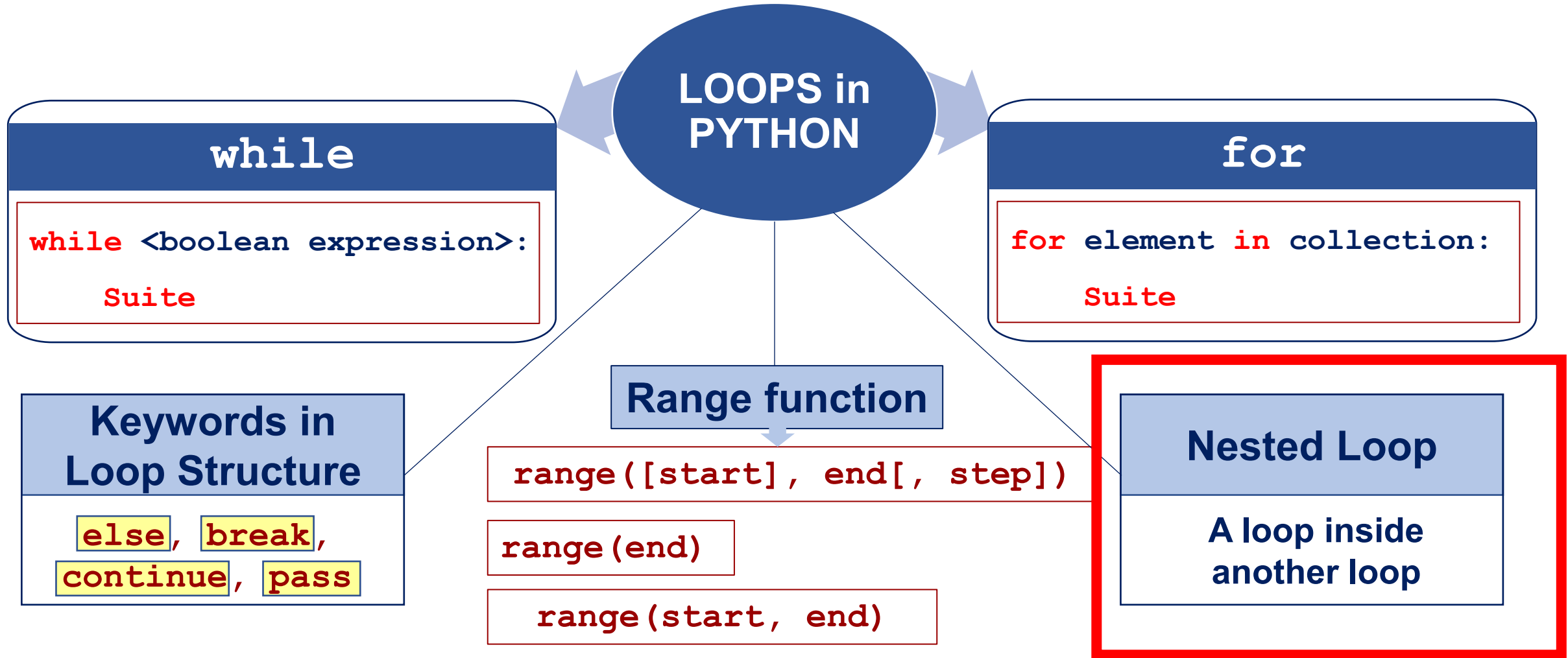
```
for num in range(4):  
    if num == 4:  
        break  
    else:  
        print(num, end = " ")  
else:  
    print("else")
```

- A. 4 else
- B. 0 1 2 3 4
- C. 1 2 3 else
- ✓ D. 0 1 2 3 else
- E. 0 1 2 3
- F. None of above

What is the output of the following code?

```
for num in range(10, -2, -2):  
    if num == 8:  
        continue  
    if num == 2:  
        break  
    print(num, end =", ")  
else:  
    print("else")
```

- A. 10, 6, 4, 0, else
- B. 10, 8, 6, 4, 0
- C. 10, 8, 6, 4,
- D. 10, 6, 4, else
- ✓ E. 10, 6, 4,
- F. None of above



Nested Loop

a loop inside another loop

- Just as it is possible to have if statements nested within other if statements, a loop may appear inside another loop.
 - An outer loop may enclose an inner loop.



What can be nested?

- Nest as many levels of loops as the system allows.
- Nest different types of loops.





Nested Loops (cont'd.)

- Key points about nested loops:
 - Inner loop goes through all of its iterations for each iteration of outer loop
 - Inner loops complete their iterations faster than outer loops
 - Total number of iterations in nested loop:

$$\begin{array}{c} \text{number_iterations_inner} \\ \times \\ \text{number_iterations_outer} \end{array}$$

What is the output of the following nested loop?

```
numbers = [10, 20]
items = ["Cat", "Puppy"]

for x in numbers:
    for y in items:
        print(x, y, end =", ")
```

- ✓ A. 10 Cat, 10 Puppy, 20 Cat, 20 Puppy,
- B. 10 Cat, 20 Cat, 10 Puppy, 20 Puppy,
- C. 10 Cat, 20 Puppy,

```
count = 0
str_sentinal = input("enter a string (enter #### to stop): ")
while str_sentinal != "####":
    for letter in str_sentinal:
        if letter == 'a':
            count +=1
            break
    str_sentinal = input("enter a string (enter #### to stop): ")
print(count , "strings with letter 'a'")
```

Sample answer to tutorial 2 Q2

```
##4 rows and 5 rows
width = int(input("Please enter pattern width: "))
#display first 4 rows
#range(1,5)-->[1,2,3,4]
for i in range(1,width):
    for j in range(i):
        print('*', end='')
    print()

#display last 5 rows
#range(5,0,-1)-->[5,4,3,2,1]
for i in range(width,0,-1):
    for j in range(i):
        print('*', end='')
    print()
```

Sample answer to tutorial 2 Q4

HANDS-ON EXERCISE



```
#Practical Exercise 2b
from sense_hat import SenseHat
sense = SenseHat()
sense.set_rotation(180)
sense.show_message("This is fun!")

red_str=input("Enter the value of the red color for message:")
green_str=input("Enter the value of the green color for message:")
blue_str=input("Enter the value of the blue color for message:")

msg_r_int= int(red_str)
msg_g_int= int(green_str)
msg_b_int= int(blue_str)
msg_color= (msg_r_int,msg_g_int,msg_b_int)
print(msg_color)

red_str=input("Enter the value of the red color for background:")
green_str=input("Enter the value of the green color for background:")
blue_str=input("Enter the value of the blue color for background:")

bg_r_int= int(red_str)
bg_g_int= int(green_str)
bg_b_int= int(blue_str)
bg_color= (bg_r_int,bg_g_int,bg_b_int)
print(bg_color)

speed_str=input("Enter the value of the display speed:")
speed=float(speed_str)
print(speed)

sense.show_message("I got it!", text_colour=msg_color, \
                  back_colour=bg_color, \
                  scroll_speed=speed)
```


Coding Exercise 3b: Error checking (runtime error)

Run your **Exercise 2b** program (on your RPi) using the following sample test cases to test the robustness of your program.

- The sample test cases are as follows:

- 1) Enter “fast” while asking float speed value

Enter the value of the display speed: fast

- 2) Enter the following value while getting color values:

Enter the value of the red color for message: 4.5

Enter the value of the green color for message: 567

Enter the value of the blue color for message: blue

Enter the value of the red color for background: 345

Enter the value of the green color for background: 564

Enter the value of the blue color for background: 678

Bad input

- In general, we have assumed that the input we receive is correct (from a file, from the user).
- This is almost never true. There is always the chance that the input could be wrong.
- *"Writing Secure Code," by Howard and LeBlanc*

```
Enter the value of the red color for message:4
Enter the value of the green color for message:5
Enter the value of the blue color for message:6
(4, 5, 6)
Enter the value of the red color for background:6
Enter the value of the green color for background:7
Enter the value of the blue color for background:8
(6, 7, 8)
Enter the value of the display speed:fast
```

```
Traceback (most recent call last):
```

```
  File "E:\LF work\LF running courses\CX1103\Lab sample codes
to supervisors\Ex2b.py", line 28, in <module>
```

```
    speed=float(speed_str)
```

```
ValueError: could not convert string to float: 'fast'
```

Exceptions: Our programs should be able to handle this.

Exception: error that occurs while a program is running

- Usually causes program to abruptly halt

Traceback: error message that gives information regarding line numbers that caused the exception

- Indicates the type of exception and brief description of the error that caused exception to be raised

What is the output of the following code if input *a* for X, 3 for Y?

```
X=int(input('X=: '))
Y=int(input('Y=: '))
result = X // Y
print(result)
```

```
>>>
===== RESTART: C:\Users\asfli\Desktop\test.py =====
X=a
Traceback (most recent call last):
  File "C:\Users\asfli\Desktop\test.py", line 104, in <module>
    X=int(input('X='))
ValueError: invalid literal for int() with base 10: 'a'
>>>
```

What is the output of the following code if input 3 for X, 0 for Y?

```
X=int(input('X=: '))
Y=int(input('Y=: '))
.....
===== RESTART: C:\Users\asfli\Desktop\test.py =====
X=3
Y=0
Traceback (most recent call last):
  File "C:\Users\asfli\Desktop\test.py", line 106, in <module>
    result = X // Y
ZeroDivisionError: integer division or modulo by zero
>>> |
```

- ✓ D. ZeroDivisionError
- E. None of above

```
X_str=input('X=')
if X_str.isdigit():
    X = int(X_str)
    Y_str=input('Y=')
    if Y_str.isdigit():
        Y = int(Y_str)
        if Y != 0:
            result = X // Y
        else:
            print("Y cannot be Zero")
    else:
        print("input Y is not a valid integer")
else:
    print("input X is not a valid integer")
```

Simplest Except suite

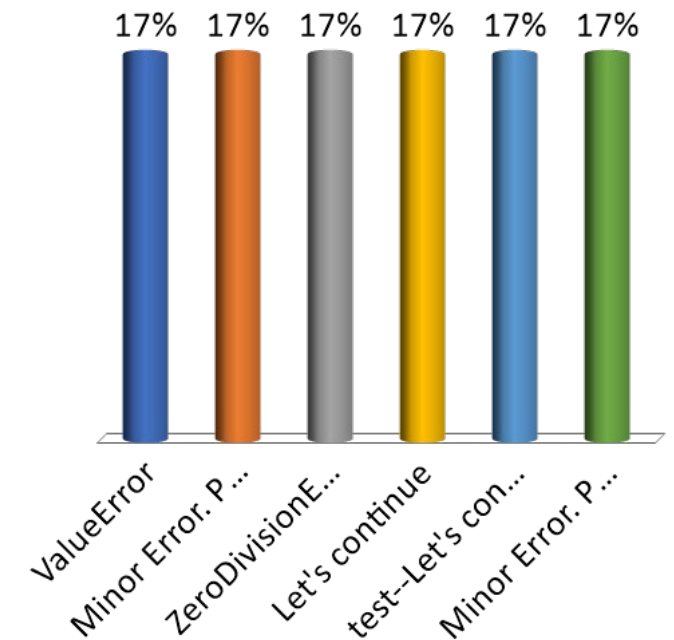





What is the output of the following code if input a for X, 0 for Y?

```
try:
    X=int(input('X='))
    Y=int(input('Y='))
    result = X // Y
    print("test")
except:
    print("Minor Error. Please ignore.", end = "--")

print("Let's continue")
```

- A. ValueError
- B. Minor Error. Please ignore.--
- C. ZeroDivisionError
- D. Let's continue
- E. test--Let's continue
- ✓ F. Minor Error. Please ignore.--Let's continue



-  8. Lab useful information_input validation, exception handling
-  9. review_while looping
-  10. review_for loop



Review/Advanced Lecture week 5-7

Availability: Item is hidden from students. It was last availa

Enabled: Statistics Tracking

Attached Files:  [Week05_Review_Composite_Data_Types](#)
 [Week06_Review_Functions.pdf](#)  (1.831
 [Week06_Review_Functions.ppsx](#)  (877
 [Week07_Review_Recursive_Functions.pc](#)
 [Week07_Review_Recursive_Functions.pp](#)



Exception Handling.(part 1)

Enabled: Statistics Tracking



Exception Handling.(part 2)

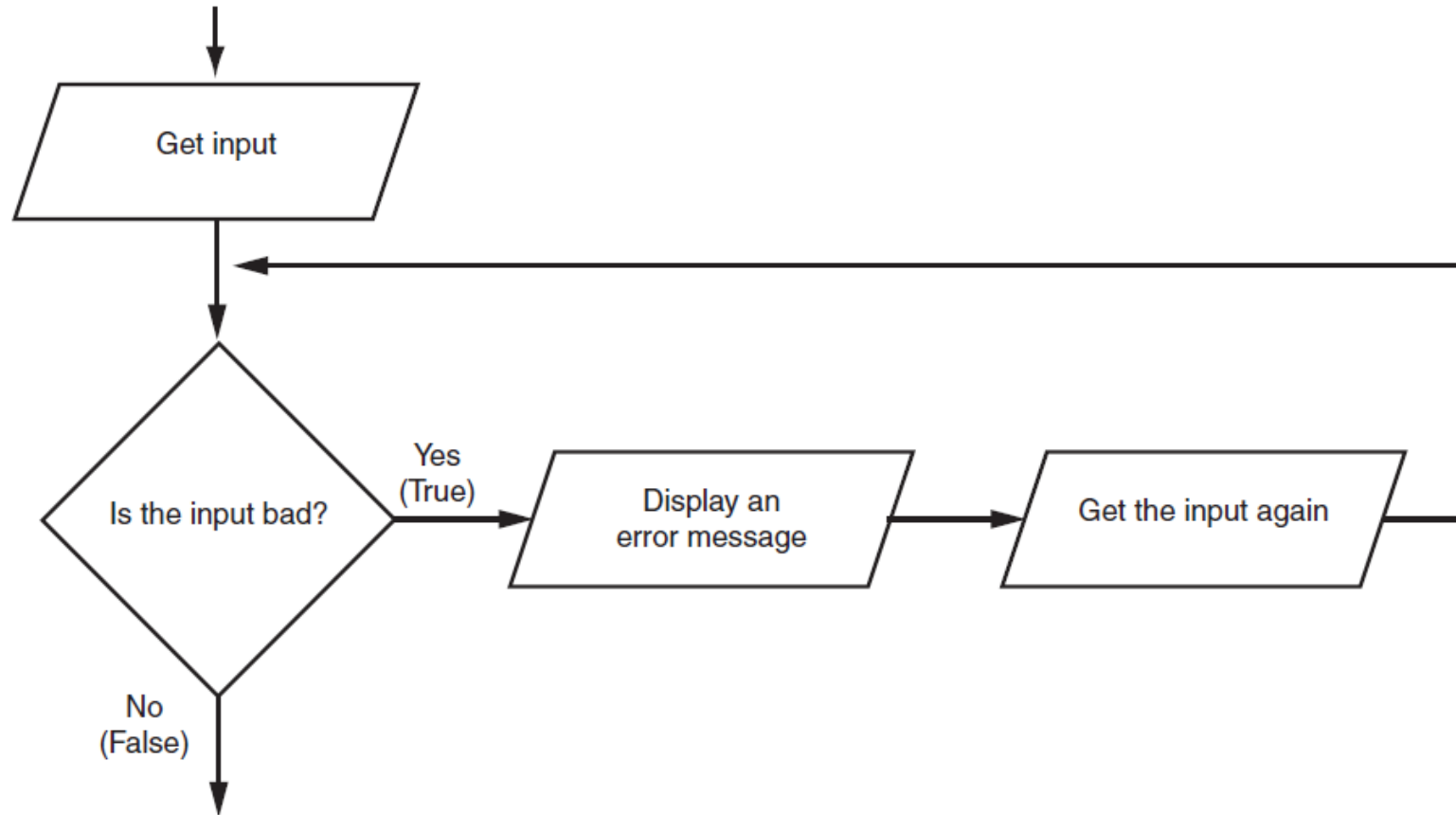
Enabled: Statistics Tracking

Input Validation Loops

- Input validation: inspecting input before it is processed by the program
 - If input is invalid, prompt user to enter correct data
 - Commonly accomplished using a `while` loop which repeats as long as the input is bad
 - If input is bad, display error message and receive another set of data
 - If input is good, continue to process the input

Input Validation Loops (cont'd.)

Figure 4-7 Logic containing an input validation loop



```
valid = False
while not valid:
    try:
        boys_num=int(input("Enter the number of boys: "))
        valid = True
    except ValueError:
        print("Invalid number, please try again.")
print("There are", boys_num, "boys in the class")
```



```
Enter the number of boys: a
Invalid number, please try again.
Enter the number of boys: b
Invalid number, please try again.
Enter the number of boys: r
Invalid number, please try again.
Enter the number of boys: t
Invalid number, please try again.
Enter the number of boys: 23
There are 23 boys in the class
```

Ex. #3 – Basic Python Programming

Invalid - use number only (1)

Enter the value of the red color for message: red

Invalid - use number only (2)

Enter the value of the red color for message: 125

Enter the value of the green color for message: green

Enter number only (1)

Enter the value of the green color for message: 345