Loop Structures

Anastasis Oulas
Evangelos Pafilis
Jacques Lagnel

Repetition

In some exercises of the previous sessions you read a certain number of numbers from the console

eg.

```
num1 = input ('Enter a number: ' )
num2 = input('Enter another number : ')
num3 = input('Enter another number : ')
```

Repetition

```
num1 = input ('Enter a number: ' )
num2 = input('Enter another number : ')
num3 = input('Enter another number : ')
num4 = input ('Enter a number: ' )
num5 = input('Enter another number : ')
```

What if more numbers are required?

Do we always have to repeat and Write endless lines of code?

Loop Structures

- They provide a solution when a series of expressions have to be repeated multiple times.

Loop Structures

- They provide a solution when a series of expressions have to be repeated multiple times.
- eg. **How can I ask the user for some numbers and calculate their sum?**

Loop Structures - Types

Two types of loop structures:

- the number of iterations (repeats) is **known** and **predefined** eg. ask for **5** numbers

→ These will be covered first

- the number of iterations is **unknown**

eg. keep on asking for numbers **until**

the user gets bored

“Pen and Paper” – write the algorithm

Read the problem (the more times the …..bester :))

It is very important to realise what has to be done

- At the start (before the loop)
- Repeatedly (while in the loop)
- At the end

Write the algorithm and describe *WHAT* needs to be done

Use the “pseudo-code” structure but don’t write code

Give an overview rather than fine detail

Calculate the Sum Example

Initialise a variable that will hold the sum (eg sum = 0)	*Do before the loop starts*
repeat 5 times	*Loop header – a condition that will control the repetitions (iterations) eg 5, 6, 7… times*
ask for a number **add it to the sum**	*Everything to be done within the loop (ie what will be repeated) is indented*
print the final sum	*Do after the loop ends*

For loop - general syntax

- **Known number** of iterations
- **for** *index* **in** range(*n*):
  - *statement(s)*
- The variable *index* will increase by value or increment of *1* for as many iterations as specified by the variable *n*.
- Its initial value is *0* (unless specified otherwise) and its final value will be *n-1*.

Calculate the Sum Example

```
sum = 0 #important to initialize sum

for i in range(5):
        number = input ('Enter a number: ' )
        sum = sum + number

print(sum)
```

Calculate the Sum Example

```
sum = 0 #important to initialize sum

for i in range(5):
        number = input ('Enter a number: ' )
        sum = sum + number

print(sum)
```

Controls the number of repetitions

Calculate the Sum Example

```
sum = 0 #important to initialize sum

for i in range(5):
   number = input ('Enter a number: ' )
   sum = sum + number

print(sum)
```

The statements that will be repeated

for loop: advanced syntax

- It is possible to let the range **start at another number**, or to specify a **different increment** (even negative; sometimes this is called the ‘**step**’):
- Examples:

“Start” value

“Stop“ value (NB: the loop stops right before the stop value)

```
for i in range(5, 10):
    print(i) #results [5, 6, 7, 8, 9]
```

Last value of “i”

“Step”

```
for i in range(0, 10, 3):
    print(i) # results [0, 3, 6, 9]
```

“Step”

```
for i in range(100,10,-20):
    print(i) # results [100, 80, 60,40,20]
```

while loop – general syntax

- **while** *condition:*
  - *statements*
- While the condition is **True** the statements in the loop are executed repeatedly. The loop **terminates** if the condition becomes **False.**
- The condition of the while loop is checked at the start of the loop.
- a while loop will not execute if initially the condition is False.

Calculate the Sum Example the "counter" approach

```
sum = 0 #important to initialize sum
counter = 0
while counter < 5:
        number = input ('Enter a number: ' )
        sum = sum + number
        counter = counter + 1
print(sum)
```

Controls whether the loop will execute or not. Checked in the beginning and after every iteration

Calculate the Sum Example the “counter” approach

```
sum = 0 #important to initialize sum
counter = 0
while counter < 5:
    number = input ('Enter a number: ' )
    sum = sum + number
    counter = counter + 1
print(sum)
```

The statements that will be repeated

Calculate the Sum Example
the “counter” approach

```
sum = 0 #important to initialize sum
counter = 0
while counter < 5:
    number = input ('Enter a number: ' )
    sum = sum + number
    counter = counter + 1
print(sum)
```

Iteration control mechanism

The statements that will be repeated

Loop Structures - Types

Two types of loop structures:

- the number of iterations (repeats) is **known** and **predefined**

  eg. ask for **5** numbers

- the number of iterations is **unknown**

eg. keep on asking for numbers **until the user gets bored**

→ This will be covered in the next slides

Let the User Control the Loop

The *while* loop is suitable for this purpose

ALGORITHM

```
while the user wants to do process
    do process
    ask user whether user wants to repeat
        read and process their reply
```

Calculate the Sum Example “the user decides” approach

```
Initialise sum
while the user is not bored
    ask for a number
    update sum
    ask user whether he/she is bored or not
    if no: continue asking questions
    If yes: stop the looping
print sum
```

Calculate the Sum Example the "counter" approach

```
sum = 0
userIsNotBored = True #important to initialize
while userIsNotBored:
    number = input ('Enter a number: ' )
    sum = sum + number
    reply = input ('Did you get bored ( y / n) ? ')
    if reply == 'yes' or reply == 'y':
        print ('thank you for your patience')
        userIsNotBored = False
print ('the sum is: ')
print(sum)
```

Calculate the Sum Example the "counter" approach

```
sum = 0
userIsNotBored = True #important to initialize
while userIsNotBored:
    number = input ('Enter a number: ' )
    sum = sum + number
    reply = input ('Did you get bored ( y / n) ? ')
    if reply == 'yes' or reply == 'y':
        print ('thank you for your patience')
        userIsNotBored = False
print ('the sum is: ')
print(sum)
```

Remember the 1st session: boolean variables :)

Looping will stop when this condition becomes false

Calculate the Sum Example the “counter” approach

```
sum = 0
userIsNotBored = True #important to initialize
while userIsNotBored:
    number = input ('Enter a number: ' )
    sum = sum + number
    reply = input ('Did you get bored ( y / n) ? ')
    if reply == 'yes' or reply == 'y':
        print ('thank you for your patience')
        userIsNotBored = False
print ('the sum is: ')
print(sum)
```

The statements that will be repeated

Calculate the Sum Example the "counter" approach

```
sum = 0
userIsNotBored = True #important to initialize
while userIsNotBored:
        number = input ('Enter a number: ' )
        sum = sum + number
        reply = input ('Did you get bored ( y / n) ? ')
        if reply == 'yes' or reply == 'y':
                print ('thank you for your patience')
                userIsNotBored = False
print ('the sum is: ')
print(sum)
```

Remember the previous session: conditional statement :)

Calculate the Sum Example the “counter” approach

```
sum = 0
userIsNotBored = True #important to initialize
while userIsNotBored:
        number = input ('Enter a number: ' )
        sum = sum + number
        reply = input ('Did you get bored ( y / n) ? ')
        if reply == 'yes' or reply == 'y':
                print ('thank you for your patience')
                userIsNotBored = False
print ('the sum is: ')
print(sum)
```

Iteration control mechanism

Calculate the Sum Example the “counter” approach

```
while userIsNotBored:
    number = input ('Enter a number: ' )
    sum = sum + number
    reply = input ('Did you get bored ( y / n) ? ')
    if reply == 'no' or reply == 'n':
        print ('ok lets go the the next round')
    elif reply == 'yes' or reply == 'y‘:
        print ('thank you for your patience')
         userIsNotBored = False
    else:
        print ('I did not understand your reply, lets go to the next round')
print(sum)
```

If you want, you may handle more responses

Nested loops

- Like the conditional statements also the loop structures can be nested

Nested Loops - pseudo-code

- Ask the user for 3 numbers each time until the user gets bored. Print the sum of each triplet and in the end print the overall sum

```
Initialise overall sum
assume user wants to continue
while user wants to continue
    Initialise triplet sum
    repeat 3 times
        ask for a number
        update triplet and overall sum
    print triplet sum
    ask user if they wish to continue
    process their reply
Print overall sum
```

Nested Loops – the code I

```
overallSum = 0 #important to initialize sum
userIsNotBored = True
while userIsNotBored:
    tripletSum = 0
    for index in range(3):
        number = input ('Enter a number: ' )
        tripletSum = tripletSum + number
        overallSum = overallSum + number
    print ('the triplet sum is: ')
    print (tripletSum)
```

→ continued in the next slide

Nested Loops – the code II

→ continued here

```
    reply = input ('Did you get bored ( y / n) ? ')
    if reply == 'yes' or reply == 'y':
        print ('thank you for your patience')
        userIsNotBored = False

print ('the overall sum is: ')
print( overallSum )
```