

Control Structures (Repetition Statement)

Python Programming
CT108-3-1-PYP

Introduction to Repetition Structures

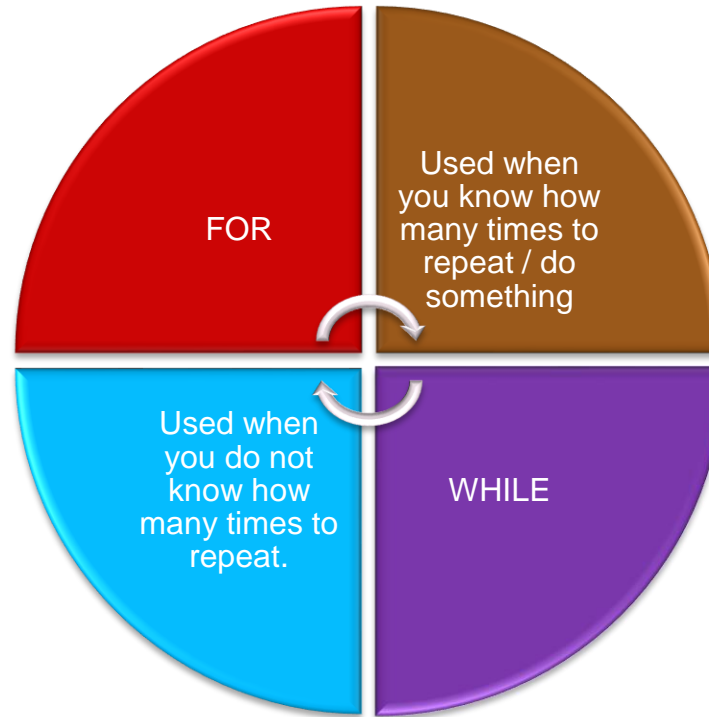
A **repetition structure**, also known as a loop or iteration, is a programming construct that allows a set of instructions or code blocks to be executed multiple times. The loop continues to repeat as long as a certain condition is met.

There are two broad categories of loops:

1.Condition-controlled loops: These loops repeat as long as a condition is true. The loop's continuation depends on the evaluation of a boolean expression (true/false). In Python, the while statement is typically used for condition-controlled loops.

2.Count-controlled loops: These loops repeat a specific number of times. The number of repetitions is known before the loop starts. In Python, the for statement is typically used for count-controlled loops, especially when iterating over a sequence (like a list, tuple, or range).

Repetition Structure in Python

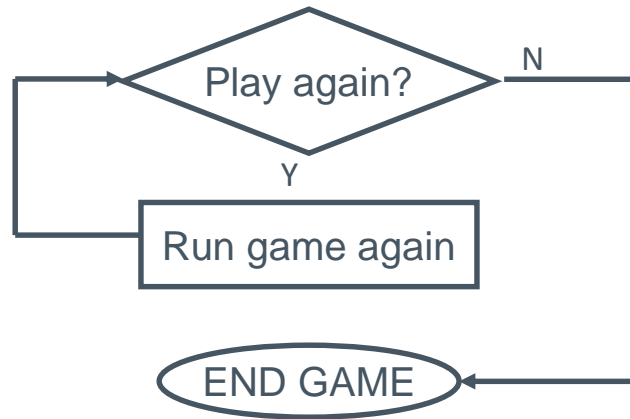


π

How To Determine If Loops Can Be Applied

- › Something needs to occur multiple times (generally it will repeat itself as long as some condition has been met).

Flowchart



Pseudo code

While the player wants to play
Run the game again

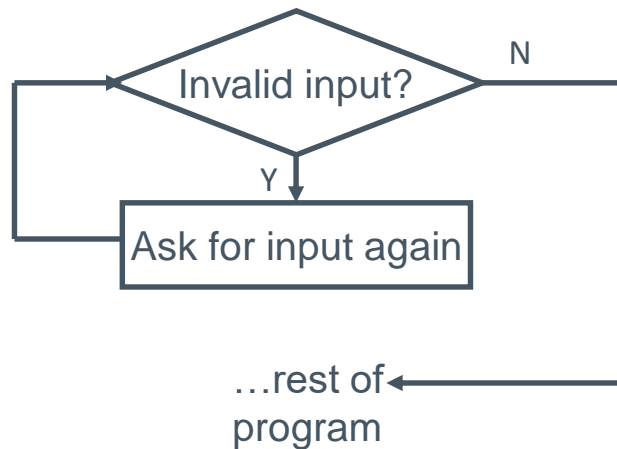
π

How To Determine If Loops Can Be

```
Enter your age (must be non-negative): -1
Enter your age (must be non-negative): 27
Enter your gender (m/f): █
```

Re-running specific parts of the program

Flowchart



Pseudo code

While input is invalid
 Prompt user for input

Types Of Loops

- › Pre-test loops
- › Post-test loops

π Pre-test loops

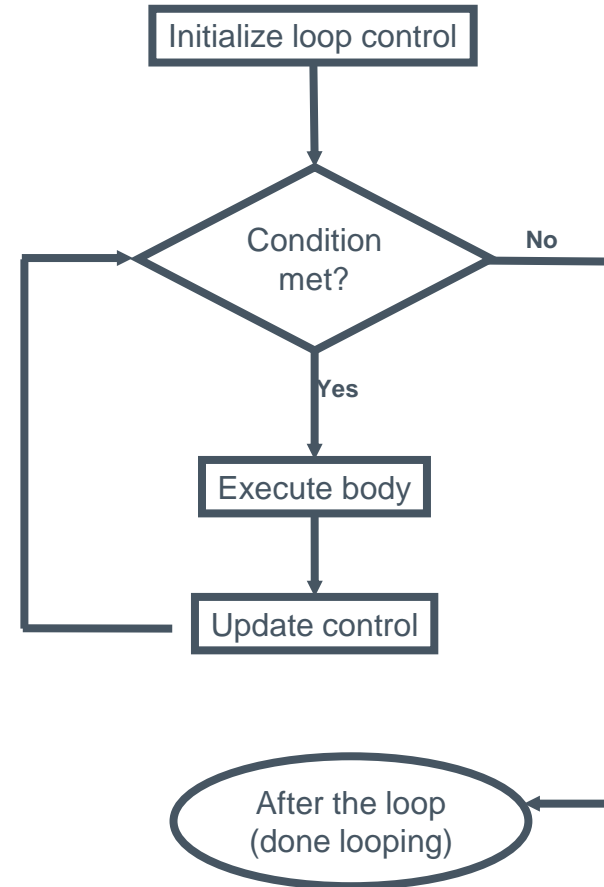
Characteristics

- › Checks the stopping condition *before* executing the body of the loop.
- › The loop executes *zero or more* times.
- › The loop may not execute at all if the condition is false initially.



Pre-test loops

1. Initialize loop control
2. Check if the repeating condition has been met
 - a. If it's been met, then go to Step 3
 - b. If it hasn't been met, then the loop ends
3. Execute the body of the loop (the part to be repeated)
4. Update the loop control
5. Go to step 2



Post-test loops

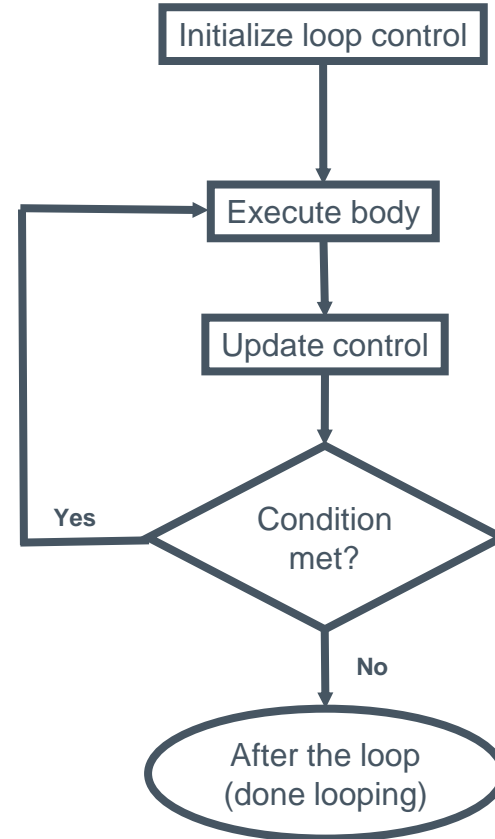
Characteristics:

- › Checking the stopping condition *after* executing the body of the loop.
- › The loop executes *one or more* times.
- › The loop executes at least once, even if the condition is false initially.

π

Post-Test Loops (**Not Implemented In Python**)

1. Initialize loop control (sometimes not needed because initialization occurs when the control is updated)
2. Execute the body of the loop (the part to be repeated)
3. Update the loop control
4. Check if the repetition condition has been met
 - a. If the condition has been met, then go through the loop again (go to Step 2)
 - b. If the condition hasn't been met, then the loop ends.



π Post-Loops In Python

- ›Note: this type of looping construct has not been implemented with this language (**Python**).
- ›But many other languages do implement post test loops.

Characteristics:

- The stopping condition is checked *after* the body executes.
- These types of loops execute one or more times.

π The While Loop

- › This type of loop can be used if it's *not known* in advance how many times that the loop will repeat (most powerful type of loop, any other type of loop can be simulated with a while loop).
- › It can repeat so long as some arbitrary condition holds true.
- › While loops are called "indefinite loops" because they keep going until a logical condition becomes False

Syntax:

```
while (Boolean expression):  
    body
```

π The while Loop – Examples

› Program 1

```
i = 1
while (i <= 3):
    print("i =", i)
    i = i + 1
print("Done!")
```

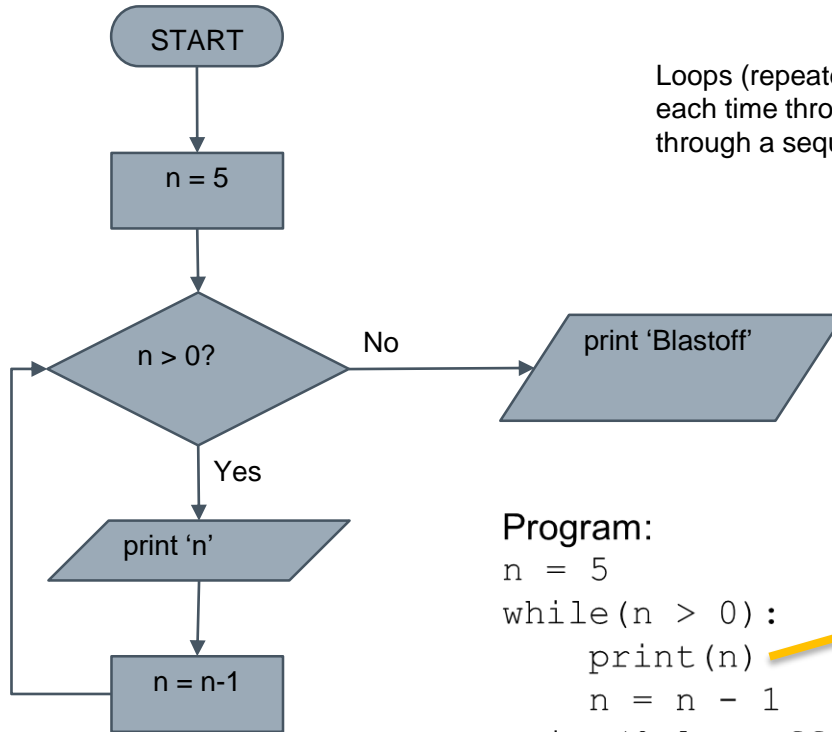
› Program 2

```
i = 3
while (i >= 1):
    print("i =", i)
    i = i - 1
print("Done!")
```



Repetition Steps or Flow - Example

Loops (repeated steps) have iteration variables that change each time through a loop. Often these iteration variables go through a sequence of numbers.



Program:

```

n = 5
while(n > 0):
    print(n)
    n = n - 1
print('Blastoff!')
  
```

Output:

```

5
4
3
2
1
Blastoff!
  
```

Repetition Steps or Flow - Example

Loops often involve **iteration variables** (also known as loop counters or control variables) that change with each iteration of the loop.

These variables are crucial because they help control the flow of the loop and track how many times the loop has executed.

In many cases, iteration variables go through a **sequence of numbers** or follow a specific pattern.

π Practice Exercise – While Loops

- › The following program that prompts for and displays the user's age.
- › Modifications:
 - As long as the user enters a negative age the program will continue prompting for age.
 - After a valid age has been entered then stop the prompts and display the age.

```
age = int(input("Age: "))  
print(age)
```

```
Age: -1  
Age cannot be negative  
Age: -123  
Age cannot be negative  
Age: 27  
You are 27 years young :P
```




Definite Loops

- › The number of iterations (repetitions) is predetermined.
- › Execute a specific number of times
- › Common structures: **for loop** (in many programming languages), where the number of iterations is set based on a range or a list.

π For Loop

Used for repeated execution of a group of statements for the desired number of times.

It iterates over the items of lists, tuples, strings, the dictionaries and other iterable objects

The For Loop – Definite

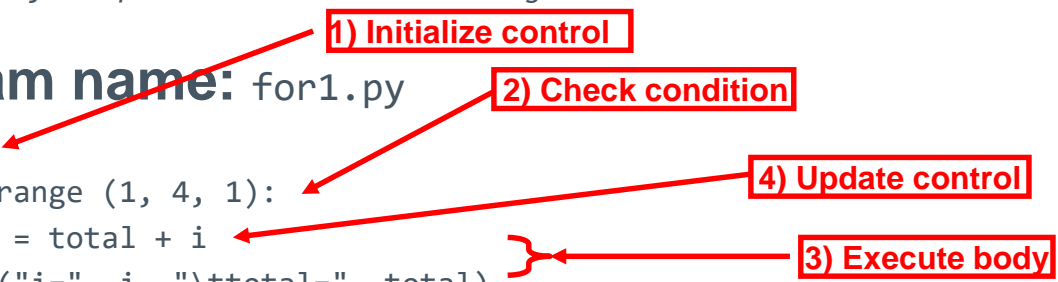
- › In Python a `for`-loop is used to step through a sequence e.g., count through a series of numbers or step through the lines in a file.

- › **Syntax:**

```
for <name of loop control> in <something that can be iterated>:
    body
```

- › **Program name:** `for1.py`

```
total = 0
for i in range (1, 4, 1):
    total = total + i
    print("i=", i, "\ttotal=", total)
print("Done!")
```



Creating a for Loop

- › To create a for loop, you start with
 - **for**,
 - followed by a *variable* for each element,
 - followed by **in**,
 - followed by the *sequence* you want to loop through,
 - followed by a **:**, and
 - finally, the *loop body*.
for *variable* **in** *sequence* :
loop body

π The for Loop

```
> i = 0
   total = 0
   for i in range (1, 4, 1):
       total = total + i
       print("i=", i, "\ttotal=", total)
   print("Done!")
```

```
i= 1      total= 1
i= 2      total= 3
i= 3      total= 6
Done!
```

π Counting Down With A For Loop

```
i = 0
total = 0
for i in range (3, 0, -1):
    total = total + i
    print("i = ", i, "\t total = ", total)
print("Done!")
```

```
i = 3    total = 3
i = 2    total = 5
i = 1    total = 6
Done!
```

π for Loop – String Example

```
word = input("Enter a word: ")  
print("\nHere's each letter in your word:")  
for letter in word:  
    print(letter)  
input("\n\nPress the enter key to exit.")
```

π Understanding for Loops

- › For strings, each element is one character.
- › Since a for loop goes through a sequence one element at a time, this loop goes through the letters in the string entered one at a time.

π for loop companion – range()

- › Can generate a sequence of numbers using range() function
- › range(10) will generate numbers from 0 to 9 (10 numbers)
- › Can also define the start, stop and step size as range (start, stop, stepsize)
- › Step size defaults to 1 if not provided.
- › Does not store all the values in memory, it would be inefficient
- › So, it remembers the start, stop, step size and generates the next number on the go

π for loop companion – range() Example

Program 1

```
•print("Counting:")  
•for i in range(10):  
    • print(i)
```

Program 2

```
•print("\n\nCounting by fives:")  
•for i in range(0, 50, 5):  
    • print(i)
```

Program 3

```
•print("\n\nCounting backwards:")  
•for i in range(10, 0, -1):  
    • print(i)  
•input("\n\nPress the enter key to exit.\n")
```



Counting Forwards

- › The first loop in the program counts forwards:

```
for i in range(10):  
    print(i)
```

- › This for loop loops through a sequence.
- › The sequence the loop moves through is created by the range() function.
- › range() function creates a sequence of numbers.
- › Give range() a positive integer and it will create a sequence starting with 0, up to, but not including, the number you gave it.

Counting by Fives

- › The next loop counts by fives:

```
for i in range(0, 50, 5):  
    print(i)
```

This is done with a call to `range()` that creates a list of numbers that are multiples of 5.

Counting Backwards

- › The following loop in the program counts backwards:

```
for i in range(10, 0, -1):  
    print(i)
```

- › This done because the last number in the range() call is -1. This tells the function to go from the start point to the end point by adding -1 each time.
- › This is the same as saying "subtract 1."
- ›

π Erroneous For Loops

- › The logic of the loop is such that the end condition has already been reached with the start condition.
- › **Example:** `for_error.py`

```
for i in range (5, 0, 1):  
    total = total + i  
    print("i = ", i, "\t total = ", total)  
print("Done!")
```

```
[csc loops 18 ]> python for_error.py  
Done!
```

Loop Increments Need Not Be Limited To One

› **While:** while_increment5.py

```
i = 0
while (i <= 100):
    print("i =", i)
    i = i + 5
print("Done!")
```

› **For:** for_increment5.py

```
for i in range (0, 105, 5):
    print("i =", i)
print("Done!")
```

```
i = 0
i = 5
i = 10
i = 15
i = 20
i = 25
i = 30
i = 35
i = 40
i = 45
i = 50
i = 55
i = 60
i = 65
i = 70
i = 75
i = 80
i = 85
i = 90
i = 95
i = 100
Done!
```

Sentinel Loops

- › A *sentinel loop* continues to process data until reaching a special value that signals the end.
- › This special value is called the *sentinel*.
- › The sentinel must be distinguishable from the data since it is not processed as part of the data.

π Sentinel Loops – Example 1

› Program name: sum.py

```
total = 0
num = 0
while(num >= 0):
    num = input ("Enter a non-negative integer (negative to end
series): ")
    num = int(num)
    if (num >= 0):
        total = total + num

print("Sum total of the series:", total)
```

π Sentinel Loops – Example 2

- › Sentinel controlled loops are frequently used in conjunction with the error checking of input.
- › Example (sentinel value is one of the valid menu selections, repeat while selection is not one of these selections)

```
selection = " "  
while selection not in ("a", "A", "r", "R", "m", "M", "q", "Q"):  
    print("Menu options")  
    print("(a)dd a new player to the game")  
    print("(r)emove a player from the game")  
    print("(m)odify player")  
    print("(q)uit game")  
    selection = input("Enter your selection: ")  
if selection not in ("a", "A", "r", "R", "m", "M", "q", "Q"):  
    print("Please enter one of 'a', 'r', 'm' or 'q' ")
```

π Using `else` Statement with `While` Loop

- › Python supports to have an `else` statement associated with a loop statement.
- › If the `else` statement is used with a `while` loop, the `else` statement is executed when the condition becomes false. Syntax

```
while expr:
```

```
    statement(s)
```

```
else:
```

```
    additional_statement(s)
```

π Using else Statement with While Loop

```
count = 0
```

```
while count < 5:
```

```
    print(count, " is less than 5")
```

```
    count = count + 1
```

```
else:
```

```
    print(count, " is not less than 5")
```

Using Condition Statement with While Loop – Example Find Even Odd Number

```
number = 2
while number < 10:
    #Find the mod of 2
    if number%2 == 0:
        print("The number "+str(number)+ " is even")
    else:
        print("The number " +str(number)+ " is odd")
    #Increment number by 1
    number = number+1
```

Nested Loops

- › One loop executes inside of another loop(s).

- › **Example structure:**

Outer loop (runs n times)

Inner loop (runs m times)

Body of inner loop (runs n x m times)

- › Program name: nested.py

```
i = 1
while (i <= 2):
    j = 1
    while (j <= 3):
        print("i = ", i, " j = ", j)
        j = j + 1
    i = i + 1
print("Done!")
```

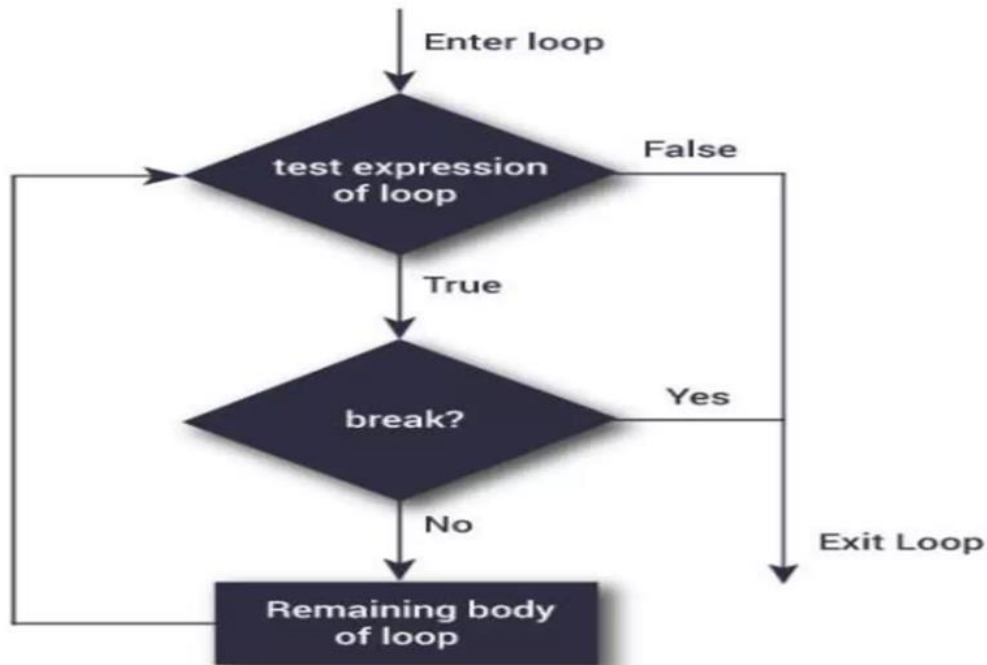
```
i = 1 j = 1
i = 1 j = 2
i = 1 j = 3
i = 2 j = 1
i = 2 j = 2
i = 2 j = 3
Done!
```

π Python break statement

- › Terminates the loop containing it
- › Control of the program flows to the statement immediately after the body of the loop.
- › If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop
- › Syntax:
 - `break`

π

Flowchart of break



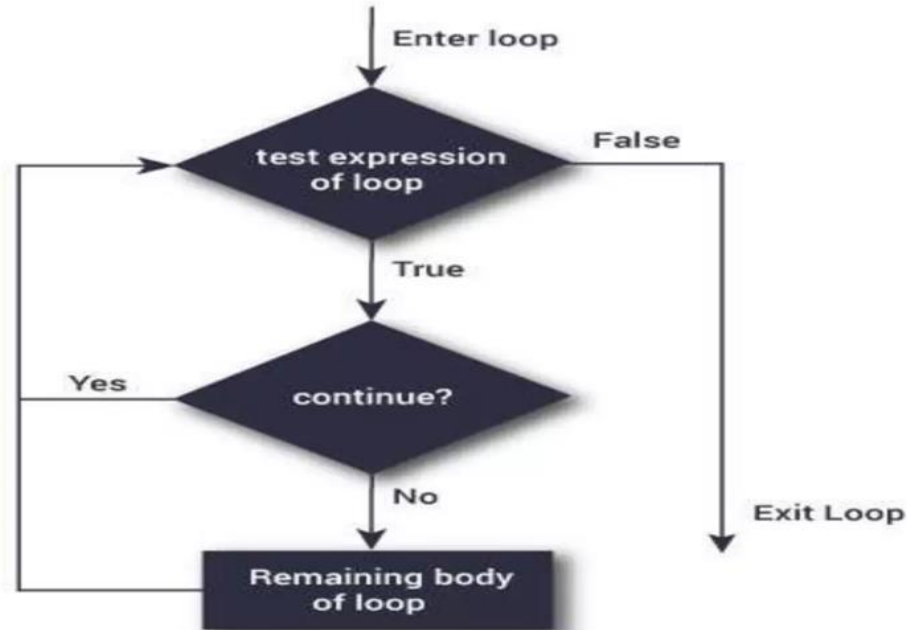
π break- Example

```
for val in "string":  
    if val == "i":  
        break  
    print(val)  
print("The end")
```

π Continue Statement

- › Is used to skip the rest of the code inside a loop for the current iteration only
- › Loop does not terminate but continues with the next iteration
- › Syntax
continue

π Flowchart of continue

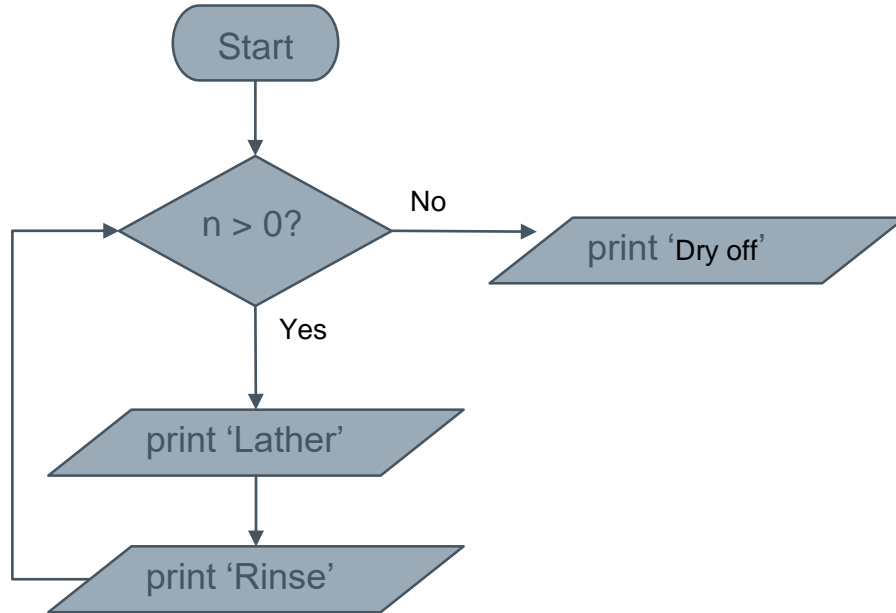


Finishing an Iteration with continue

```
for val in "string":  
    if val == "i":  
        continue  
  
    print(val)  
print("The end")
```

π

Test Yourself!



```
n = 5
while n > 0 :
    print ('Lather')
    print ('Rinse')
print ('Dry off!')
```

Common Mistakes: While Loops

- › Forgetting to include the basic parts of a loop.
 - Updating the control - Iteration

```
i = 1
```

```
while(i <= 4):
```

```
    print("i =", i)
```

```
i = 1
i = 1
i = 1
i = 1
i = 1
i = 1
i = 1
i = 1
i = 1
i = 1
```

π Infinite Loop

Loops must contain within themselves a way to terminate

- Something inside a while loop must eventually make the condition false

Infinite loop: loop that does not have away of stopping

- Repeats until program is interrupted
- Occurs when programmer forgets to include stopping code in the loop

Infinite Loops

- › Infinite loops never end (the stopping condition is never met).
- › They can be caused by logical errors:
 - The loop control is never updated (Example 1 – below).
 - The updating of the loop control never brings it closer to the stopping condition (Example 2 – next slide).

- › **Example 1:** infinite1.py

```
i = 1
while (i <= 10):
    print("i = ", i)
    i = i + 1
```

```
i = 1
i = 1
i = 1
i = 1
i = 1
i = 1
i = 1
i = 1
i = 1
i = 1
```


π Infinite Loops

› **Example 2:** infinite2.py

```
i = 10  
while (i > 0):  
    print("i = ", i)  
    i = i + 1  
print("Done!")
```

```
i = 14477  
i = 14478  
i = 14479  
i = 14480  
i = 14481  
i = 14482  
i = 14483
```