ENGR 1330
Computational Thinking with Data Science

4. Programming Principles - Loops

Turgut Batuhan Baturalp, (Dr. Batu)
Whitacre College of Engineering
Texas Tech University

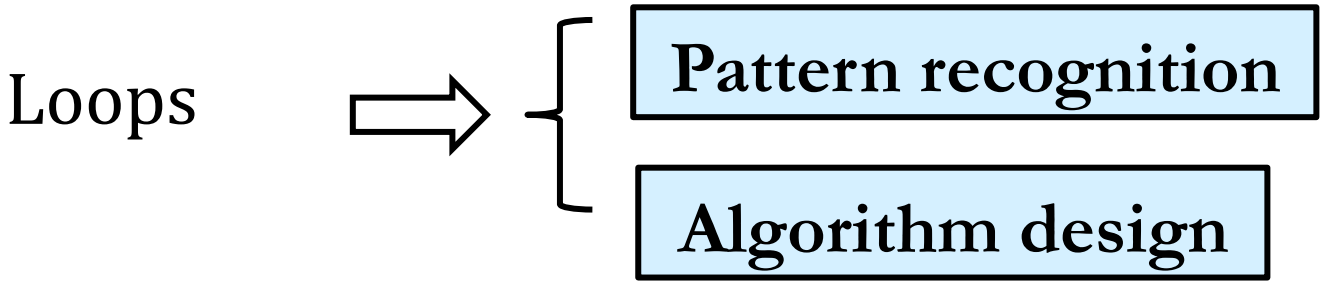
Topic Outline

- Loops in Python
  - For Loop
  - While Loop
  - Nested Loops
  - Conditional Statements of Loops

Objectives

- To understand concept of loops.
- To understand loops types available in Python.
- To understand and implement loops in various examples and configurations.

Computational Thinking Concepts



What is a Loop in Coding?

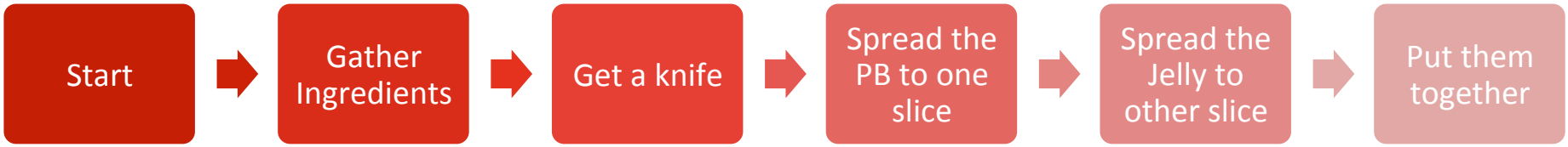
Loops: The action of doing something over and over again.

Basically, Loops repeats a portion of code a set number of times until a process is complete.

Repetitive tasks (Loops) are very common and essential in programming. They save time in coding and minimizes the coding errors.

Example

- If you think any mass manufacturing process, we apply the same process again and again. Even for something very simple such as preparing a PB sandwich:



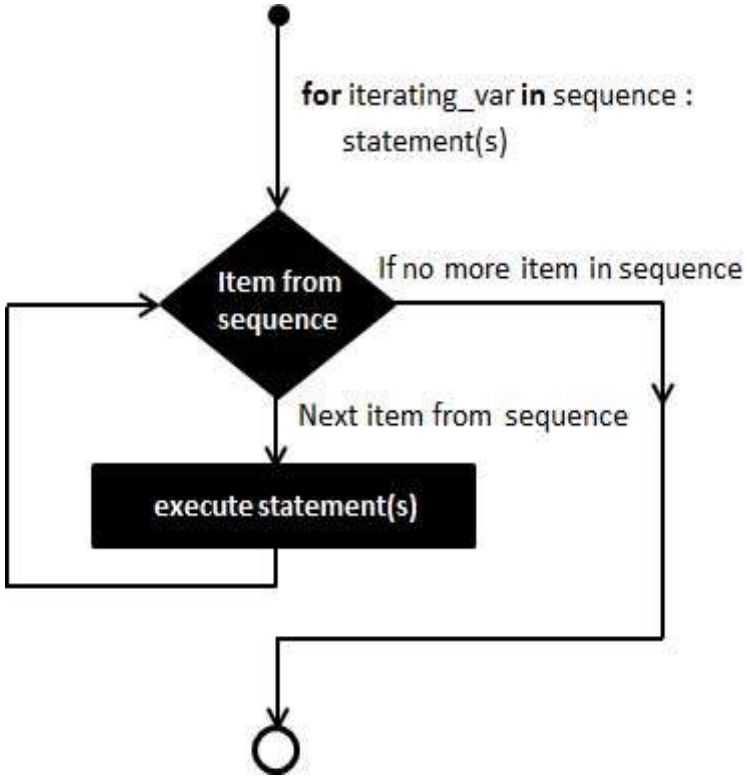
- If we need to make 1000 PB sandwich, it is much easier with an automated process such as loops.
- Also your code will be much more organized and shorter..

Types of Loops

- Since loop is a sequence of instructions that is continually repeated until a certain condition is reached. There are 2 main types loops based on the condition.

For Loops

- This flow chart explains how a for loop works:
  - If a sequence contains an expression list, it is evaluated first.
  - Then, the first item in the sequence is assigned to the iterating variable *iterating_var*.
  - Next, the statements block is executed. Each item in the list is assigned to *iterating_var*, and the statement(s) block is executed until the entire sequence is exhausted.



For loops

- Computers can repeat things over and over very quickly.
- A for loop steps through each of the items in a collection type, or any other type of object which is iterable.

```
for <item> in <collection>:
        <statements>
```

```
for <item> in <collection>:
        <statements>
else:
        <other statements>
```

- If <collection> is a list or a tuple => the loop steps through each element of the sequence

```
for c in [1, 2, 3, 4, 5]:
        print("c=", c)
```

For loops

- If <collection> is a string, then the loop steps through each character of the string.

```
for c in "Texas tech university":
        print(c)
```

- If the elements of <collection> are collections then <item> can match the structure of the elements.

```
for c in ["first", "second", "third"]:
        print(c)

for (x, y) in [(-1, 1), (-2, 2), (-3, 3)]:
        print("x=", x, ",y=", y)
```

For loops with range and enumerate functions

- range(number): returns a list of numbers from one up to but not including the number passed to it

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

- range(start, end): returns a list of numbers from start up to but not including the number end passed to it

```
for n in range(1, 10):
        print(n)
```

For loops with range and enumerate functions

- range(start, end, steps): returns a list of numbers from start up to but not including the number end with incremental step steps passed to it

```
for n in range(1, 10, 2):
        print(n)
```

- Loop with elements and indices access:

```
myList = [“one”, “two”, “three”]
for i, e in enumerate(myList):
        print(“index=“, i, “; element=“, e)
```

for loop examples

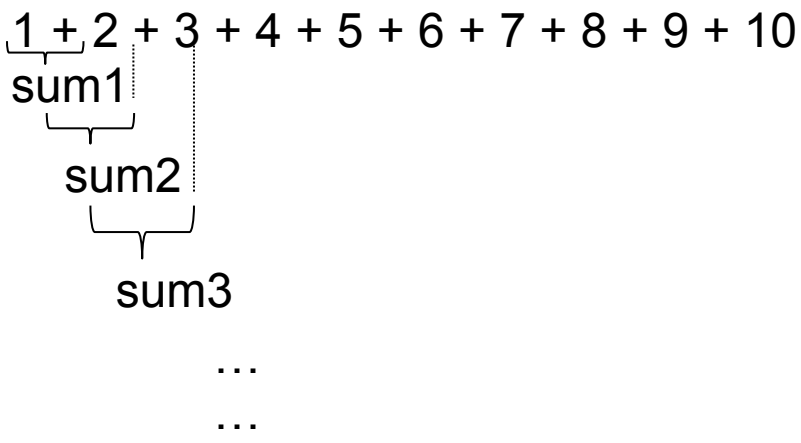
- Example1: Sum all the numbers from 1 to n

  1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10

  How to instruct computer to solve this problem?

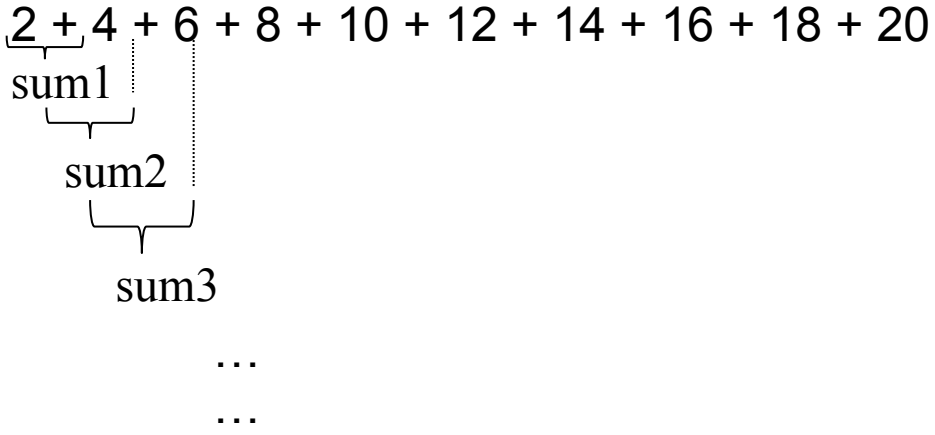
for loop examples

- Example1: Sum all the numbers from 1 to n



For loop examples

- Example2: Sum all the even numbers from 1 to n



For loop examples

- Example3: Print the triangle below. Allow user to enter the height of the triangle.

```
*
* *
* * *
* * * *
* * * * *
```

Hint: to print to screen without moving cursor to new line => use print(“*”, end=“ ”)

for loop examples: live demo

- Example1: Sum all the numbers from 1 to n

  1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10

- Example2: Sum all the even numbers from 1 to n

  2 + 4 + 6 + 8 + 10 + 12 + 14 + 16 + 18 + 20

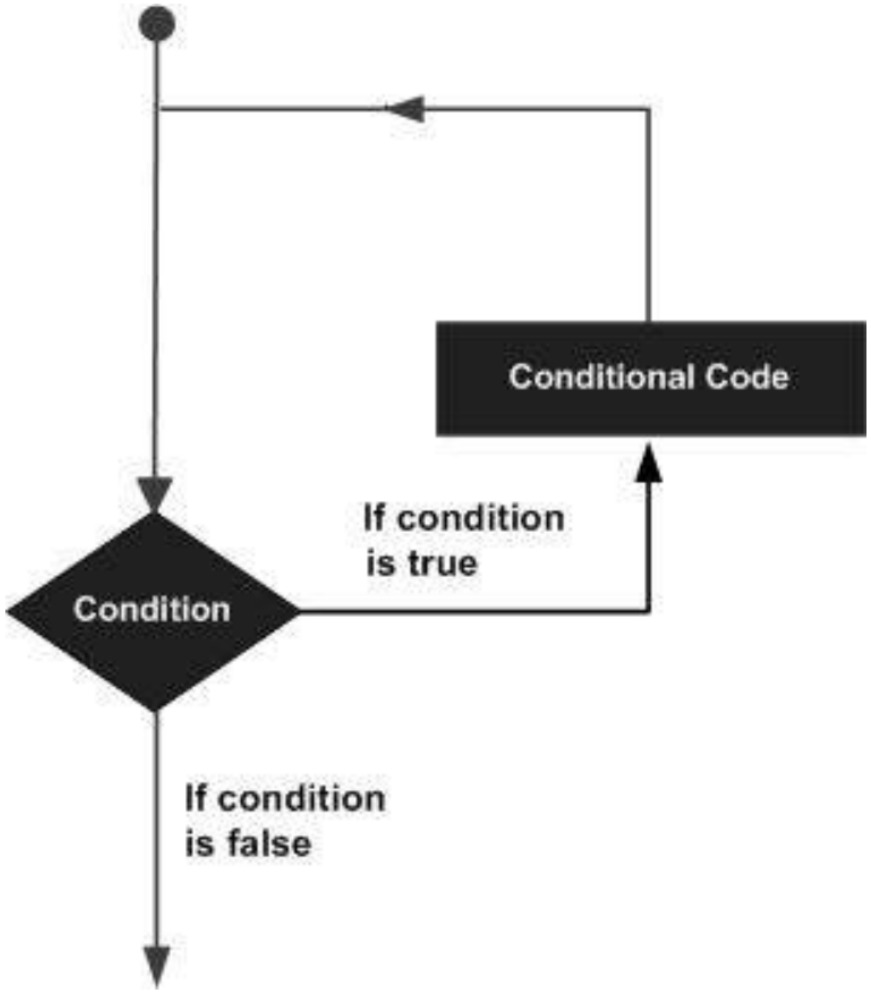
- Example3: Print the triangle below. Allow user to enter the height of the triangle.

```
*
* *
* * *
* * * *
* * * * *
```

Hint: to print to screen without moving cursor to new line => use print("*", end=" ")

While Loops

- This flow chart explains how a while loop works:
  - The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.
  - When the condition becomes false, program control passes to the line immediately following the loop.



while loops

- Use when a loop is required with respect to a conditional expression

```
while <boolean expression>:
                    <statements>
```

When do NOT want to execute statements in case of a false boolean expression

```
while <boolean expression>:
                    <statements>
else:
                    <other statements>
```

while loops

- Use when a loop is required with respect to a conditional expression

```
while <boolean expression>:
        <statements>
```

```
while <boolean expression>:
        <statements>
else:
        <other statements>
```

When we want to execute statements in case of a false boolean expression

while loop examples: live demo

- Example1: Sum all the numbers from 1 to n

  1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10

- Example2: Sum all the even numbers from 1 to n

  2 + 4 + 6 + 8 + 10 + 12 + 14 + 16 + 18 + 20

- Example3: Print the triangle below. Allow user to enter the height of the triangle.

```
*
* *
* * *
* * * *
* * * * *
```

Hint: to print to screen without moving cursor to new line => use print("*", end=" ")

Nested Loops

- Python programming language allows to use one loop inside another loop.
- You can mix and match for or while loops and make them nested into each other.

- A loop that occurs within another loop. Inner loop will be executed one time for each iteration of outer loop.

```
for <item1> in <collection1>:
        for <item2> in <collection2>:
                <statements>
```

- Print multiple tables from 1 to 10

```
for n in range(1, 11):
        for m in range(1, 11):
                print(n, ”x”, m,”=“, n*m)
```

Control Statements of Loops

- **break** statement: Terminates the loop statement and transfers execution to the statement immediately following the loop.
- **continue** statement: Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
- **pass** statement: The pass statement is used when a statement is required syntactically (for the syntax) but you do not want any command or code to execute.