What are loops?

- A loop is a control structure that allows a program to execute one or more statements repeatedly so long as certain condition is satisfied
- Python provides
  - the *while* statement for loops
  - the *for* statement for loops

While loop: an example

- Suppose we want a program to display on the screen the numbers from 1 to 100
- Of course, we don't want to write a program with 100 print statements

While loop: an example

- Instead, you would want a program that works like this:

1. Start with n = 1
2. Print n
3. Increase n by 1
4. Repeat steps 2 and 3 until 100 is displayed

While **loop**

While loop: an example

- This program allows us to do precisely that

```
n = 1
while n <= 100:
    print(n)
    n = n + 1
```

While loop: an example

- What the program says

```
n = 1
while n <= 100:
    print(n)
    n = n + 1
```

start with

while this is true

do this

While loop: an example

- Run the program, we get

```
1
2
3
...
100
```

While loop: How it works?

loop condition

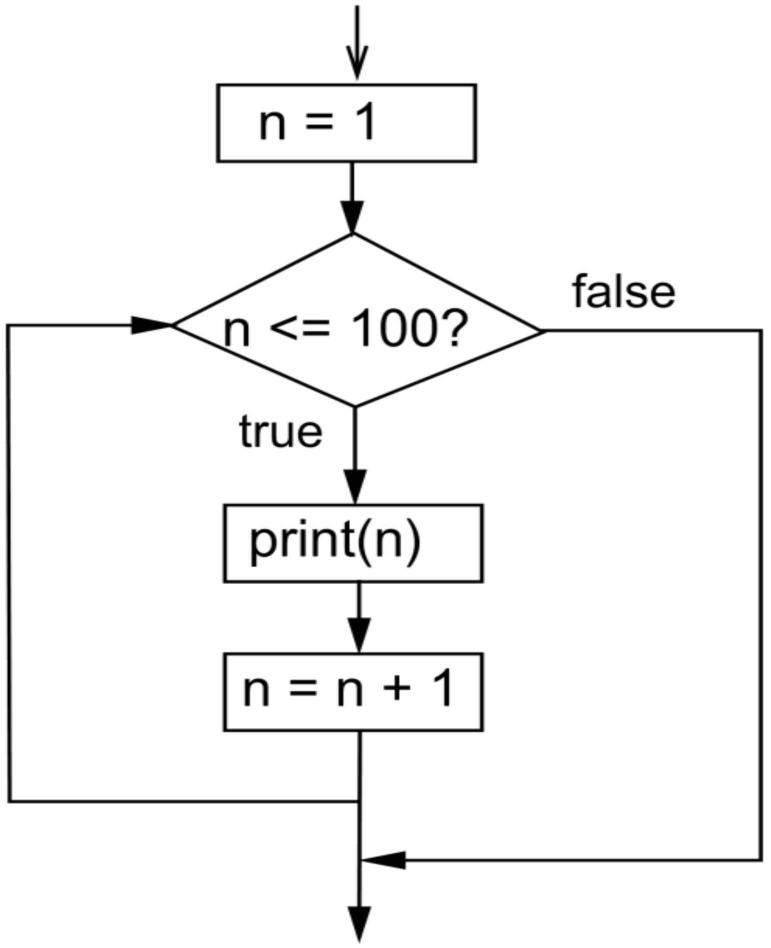
```
n = 1

while n <= 100:

        print(n)

        n = n + 1
```

loop body

- Statements in loop body are executed so long as loop condition is true
- Each execution of the loop body is called an iteration

While loop: Flow of control

```
n = 1
while n <= 100:
        print(n)
        n = n + 1
```



While loop: General syntax

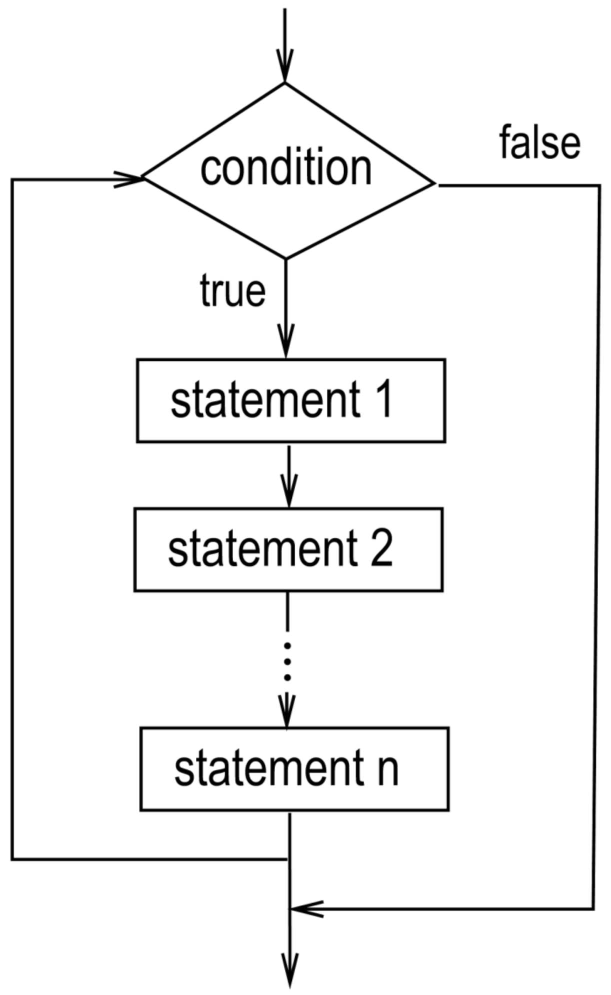
loop condition

```
while <condition>:
   <statement 1>
   <statement 2>
   ...
   <statement n>
```

loop body

While loop: Flow of control (behavior)

```
while <condition>:
    <statement 1>
    <statement 2>
    …
    <statement n>
```



While loop: Example 1

- **Problem:**

  Write a program to display even numbers from 2 to 10, inclusive

One approach

- Start with program that displays numbers from 1 to 5 (instead 100)

```
    n = 1
  while n <= 5:
        print(n)
        n = n + 1
```

- How to change it for our new problem?

One approach

```
n = 1

while n <= 5:

     print(n)

     n = n + 1
```

- How to change it for our new problem?

1. The first number we want to print is 2. So initialize n to 2
2. Once a number n is printed, the next number to consider is n+2. So change n+1 to n+2
3. The last number to print is 10. So change n <= 5 to n

One approach

```
n = 2 ~~1~~

while n <= 10 ~~5~~:

        print(n)

        n = n + 2 ~~n + 1~~
```

$\longrightarrow$

```
n = 2
while n <= 10:
        print(n)
        n = n + 2
```

Example 2

- Problem: Modify previous program to display even numbers from 10 down to 2, inclusive
- Changes?

1. The first number we want to print is 10. So initialize n to 10
2. Once a number n is printed, the next number to consider is n-2. So we “update” n with n = n-2
3. The last number to print is 2. So loop condition is n >= 2

Example 2

```
n = ~~2~~  10
while ~~n <= 10~~ n >= 2:
        print(n)
        n = ~~n + 2~~  n - 2
```

$\longrightarrow$

```
n = 10
while n >= 2:
        print(n)
        n = n - 2
```

Example 3

- Program to say “Hello” until the user wants to stop
- Sample run

```
Hello!
Continue? (y/n): y
Hello!
Continue? (y/n): y
Hello!
Continue? (y/n): n
```

Example 3

- Key question (for this problem):
  - How to control the loop?
  - What can be chosen as the control condition?

Example 3

Suppose we have this idea:

- We will need to get the response from the user
- So let us use this as the loop condition

```
userResponse == “y”
```

- Program structure

```
...
while userResponse == “y”:
    ....
```

Example 3

- Issues

```
initialize?
while userResponse == “y”:
        actions?
```

- Initialization?

```
    userResponse == “y”
```

- Actions?
  - print “Hello”
  - ask if user wants to continue

Example 3: Code

```
response = "y"
while responses == "y":
    print("Hello!")
    response = input("Continue? (y/n): ")
```

Version 2

- Not uncommon to use a boolean variable as the loop condition
- Suppose we call our variable **`keepLooping`**
- Program structure

```
...
while keepLooping:
    ...
```

Code

```
???
while keepLooping:
    ???
```

- In the loop body
  - Print “Hello”
  - Ask for user response
  - Update **`keepLooping`**
- Initialize **`keepLooping`** to **`True`**

Code

```
keepLooping = True
while keepLooping:
        print("Hello")
        userResponse = input("Continue?(y/n): ")
        keepLooping = userResponse == "y“
```

Version 3

- Use boolean value True as loop condition
- Loop structure:

```
while True:
    ...
```

- How can we get out of the loop?
- Use the `break` statement

Version 3

- General structure

```
while True:
    ...
    if ...:
      break
```

Version 3

```
while True:
    ???
```

- For each iteration,
  - Print "Hello"
  - Get user's response
  - If user wants to stop, issue the break

Code

```
while True:
        print("Hello")
        userResponse = input("Continue?(y/n): ")
    if userResponse != "y":
            break
```

For **loop**

What is the *for* loop?

- Essentially, this is what we can do with for loop
- We can iterate over the elements of a collection and execute a group of statements for each of them
- We can visit each of the elements and perform some actions for each

for loop: an example

- Suppose we have this list

```
numbers = [10, 20, 30, 40]
```

- We want to display the elements, each on a separate line
- We want to visit each element and display that element on the screen

for loop: an example

- We can use this for loop to do that

```
for n in numbers:
    print(n)
```

Code

```
# Define a list of numbers
numbers = [10, 20, 30, 40]
# Display the numbers in the list one by one
for n in numbers:
        print(n)      # line 6
```

for loop: How it works

- The loop allows us to **visit** the elements of the list
- The first time the loop iterates, the first element (10) is assigned to variable n, then the statement on line 6 is executed, which prints 10 on the screen
- The second time the loop iterates, the **next value (20)** is assigned to n, and then 20 displayed by the statement on line 6
- This process continues for the remaining elements.
- The overall result is that the four numbers in the loop are displayed on the screen.

Notes: What you need to know about lists, for now?

- Define a list

  mylist = [“dog”, “cat”, “fish”, “bird”]

- Get the length of a list, i.e. number of elements in the list

  len(mylist)

- Get a particular element by its index

  mylist[0], mylist[1], etc.

General syntax

$$\begin{tikzcd}\text{iterable collection} \arrow[d] \\ \text{numbers}\end{tikzcd}$$

```
for n in numbers:
    print(n)
```

- An iterable collection is one whose elements can be iterated over by a for loop
- It is an object that can be passed to method **iter**

General syntax

loop variable, representing an element in the collection
for a particular iteration

```
for <variable> in <iterable collection>:

    <statement 1>

    <statement 2>

    ...

    <statement n>
```

loop body
statements in loop body can refer to loop variable

Using for loop with range objects

- Loops that iterate over a range of integer values are very common
- To simplify the creation of such loops, Python provides the range function to create range objects
- A range object is a sequence of integers that can be iterated over with a for loop
- A range object is an iterable collection

Example

```
for n in range(1, 5):
    print(n)
```

- When run it displays

```
    1
    2
    3
    4
```

- range(1, 5) generates a sequence from 1 to 4 (5 is not included)

General syntax

range function can take 1, 2 or 3 arguments

- range(end)
- range(start, end)
- range(start, end, step): where step can be positive or negative

General syntax

- range(start, end)

  generates increasing sequence from start to end-1

  e.g. range(1, 10) generates integers 1, 2, ... 9
- range(end)

  generates increasing sequence from **0**, 1, up to end-1

  e.g. range(10) generates integers **0**, 1, 2 ... 9

General syntax

- range(start, end, step) where step is positive

  generates an increasing sequence of integers

  start, start+step, start + 2 * step, ...

  which stops before reaching value end

- range(0, 10, 2) generates 0, 2, 4, 6 and 8

General syntax

- range(start, end, step) where step is negative

  generates a decreasing sequence

  start, start + step, start + 2 * step ....

  which stops before reaching value end
- e.g. range(10, 0, -2) generates 10, 8, 6, 4 and 2

Using for loop with strings

- Strings (which are sequences of characters) are also iterable collections
- We can use for loop to iterate over its characters

Example

```
for char in “Hello”:
    print(char)
```

- displays

```
H
e
l
l
o
```

Example

Problem:

- Let **marks** be a list of marks between 0 and 100
- Write a program to display all marks that are 50 or above
- Also display the percentage of pass marks (marks that are 50 or above)

Approach

- To list all the pass marks, we need to visit every mark in the list

  And if a mark is 50 or more, we print it on the screen

- How to find the pass rate?
- We need to count all the pass marks. This can be done by using a variable count, which is initially 0

  And every time we get a pass mark, we increase count by 1

Code

```
marks = [60, 50, 40, 20, 90]
count = 0
for m in marks:
    if m >= 50:
        print(m)
        count = count + 1

passRate = count / len(marks)

print("The pass rate is", round(passRate * 100), "%")
```

Example

Problem:

- Write a program to display a table with 2 columns
- The first column display n from 1 to 16
- The second column displays 2^n

Approach

- Use the range function to generate numbers from 1 to 16
- Use a for loop to visit the numbers, and for each number (denoted by n), print n and 2^n

Code

```
start = 1
end = 16
for n in range(start, end+1):
    print(n, 2**n)
```

Example - Triangle of stars

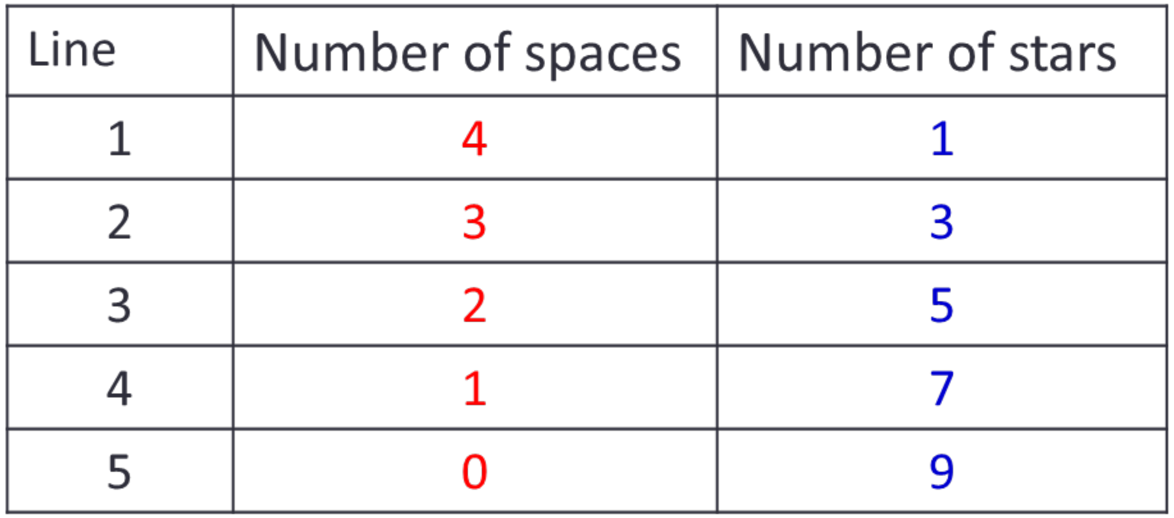
- Size = 5 (number of lines, height)

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

Write a program that reads n and displays a triangle of size n

Approach

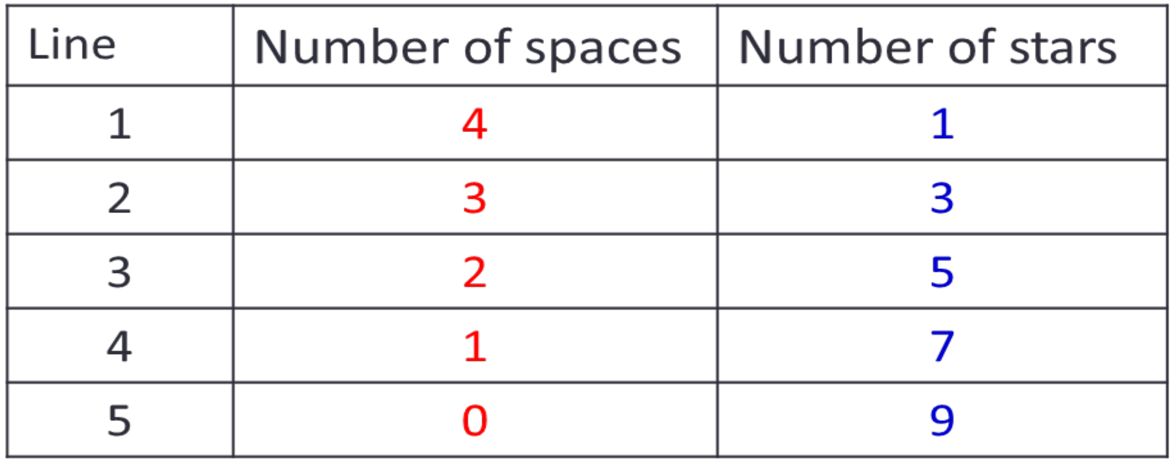
- Take size = 5
- There are 5 lines denoted by n = 1, 2, 3, 4 and 5
- Each line has a number of spaces and a number of stars



Line	Number of spaces	Number of stars
1	4	1
2	3	3
3	2	5
4	1	7
5	0	9

Approach

- Observe relationship between consecutive rows/lines



Line	Number of spaces	Number of stars
1	4	1
2	3	3
3	2	5
4	1	7
5	0	9

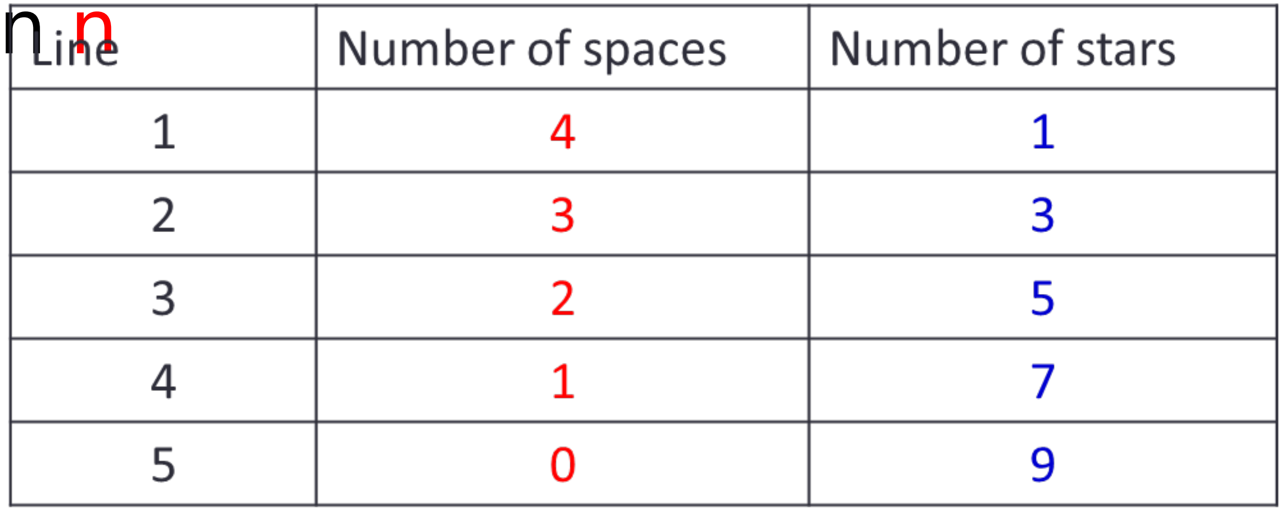
- nb of spaces of a row = nb of spaces of previous row – 1
- nb of stars of a row = nb of stars of previous row + 2
- We start with 4 spaces and 1 star

Code

```
size = 5
spaces = size-1        # for line 1
stars = 1              # for line 1
for line in range(1, size+1):
    line = "." * spaces + "*" * stars
    print(line)
    spaces = spaces  - 1 # for next line
    stars = stars + 2
```

Version 2

- Base on how number of spaces and stars of line n depend on n



Line	Number of spaces	Number of stars
1	4	1
2	3	3
3	2	5
4	1	7
5	0	9

- nb of spaces of row n = size – n
- nb of stars of row n = 2*n - 1

Code

```
size = 5
for n in range(1, size+1):
    spaces = size - n
    stars = 2*n - 1
    line = "." * spaces + "*" * stars
    print(line)
```

Acknowledgement

- Acknowledgement to https://www.py4e.com/ and to Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information.

- Also acknowledgement to Rick Skarbez (CSE1PES/CSE5CES).