Iteration with Simple Numeric Programs

CS115

INTRODUCTION TO PROGRAMMING IN PYTHON

WEEK 3

Iteration/Repetition

Repetition statements allow us to execute a statement multiple times

Often, they are referred to as *loops*

Like conditional statements, they may be controlled by Boolean expressions

Python has two kinds of repetition statements: `while` and `for` loops

The while Statement

```
while <condition>:
    < statement >
    < statement >

    ...
```

`<condition>` is a `Boolean` expression or value.

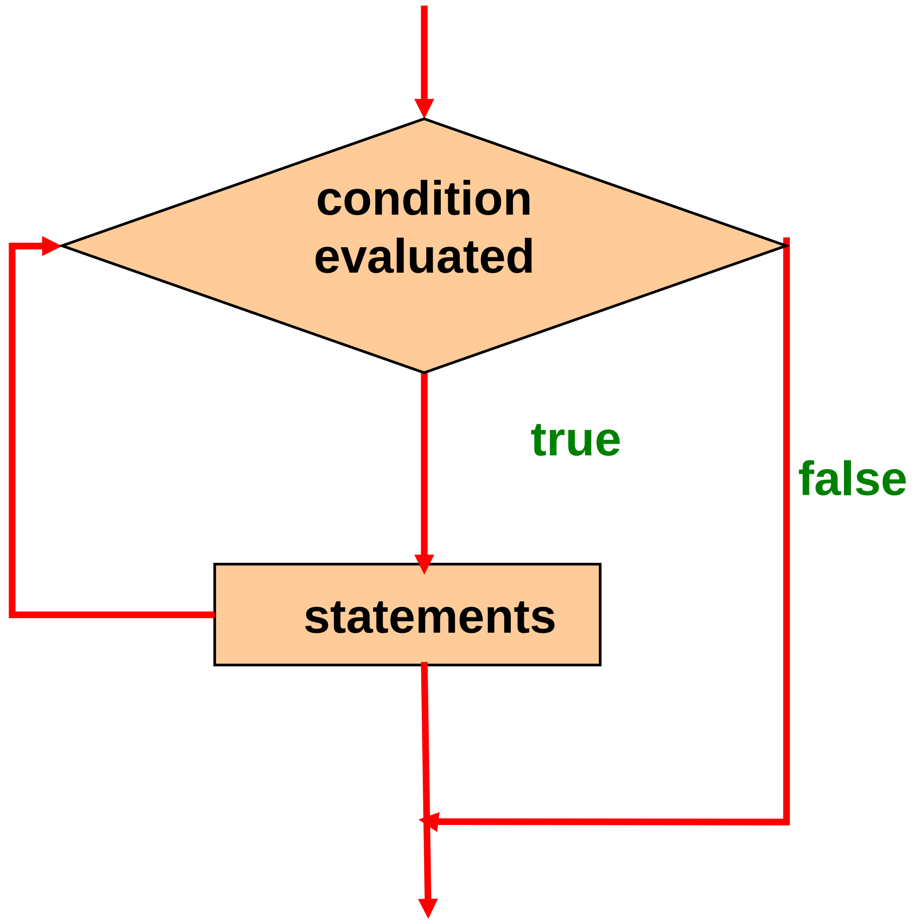
if `<condition>` is `True`, execute steps indented under the while statement.

After statements execute, while `<condition>` is again evaluated.

Repeat steps until `<condition>` is `False`

Note: like conditional statements, blocks of code in loops are indicated by indentation.

Logic of a while Loop



While Loop Example

1. Write a Python script that does the following. Assume you deposit 10000TL into a bank account, output the number of years it takes to double the original investment, assuming a yearly interest rate of 15%.
   - See: `03_investment.py`
2. In a biology experiment a microorganism population doubles every 10 hours. Write a Python program to input the initial number of microorganisms and output how long (days and remaining hours) it will take to have more than 1000000 organisms.
   - See: `03_biology.py`
3. Write a Python script that inputs positive integers until a negative value is input. The program should output the average of the values input.
   - See: `03_average.py`
4. Write a Python guessing game program. The program should generate a random int between 1 and 10. The user will try to guess the number until he/she finds it. The program should output the number of guesses made.
   - See: `03_while_game.py`

Random Package

A **Python package** is a collection of functions and methods that allows you to perform lots of actions without writing your own code.

Python includes functionality for generating random values in the `random` package.

To access the functions defined in the package, you must first import the containing package.

Syntax:

```
import random
num = random.randint(1,10)
```

OR

```
from random import *
num = randint(1,10)
```

The function `randint` generates a random value between the two values, inclusive.

To generate a random floating-point value between 0 and 1 (exclusive), you may use the `random()` function.

For Loops

Another type of loop in Python is the `for` loop, which is appropriate for definite repetition.

Everything that can be written with a for loop can be written with a while loop, but while loops can solve some problems that for loops don't address easily.

In general, use for loops when you know the number of iterations you need to do - e.g., 500 trials.

Using a for loop when possible will decrease the risk of writing an infinite loop and will generally prevent you from running into errors with incrementing counter variables.

Example: Print squares of 1 to 10

```
for i in range(1,11):
    print(i ** 2)
```

For Loops

```
for <variable> in range(<some_num>):
    < statement >
    < statement >
    ...
```

Each time through the loop, `<variable>` takes a value

The first iteration, `<variable>` starts at the smallest value

The next iteration, `<variable>` gets the previous value of the variable and increments (`+1`)

Ends when it reaches `some_num - 1`

range(start,stop,step)

The range function generates a list of numbers, generally used to iterate with for loops.

The function has 3 parameters:

- `start`: starting number of the sequence
- `stop:` generate numbers up to but not including the stopping value
- `step:` update/difference between each number in the sequence.

default values are `start = 0` and `step = 1`

loop until value is `stop - 1,` inclusive

For Loops

```
mysum = 0
for i in range(5, 11, 2):
    mysum += i
print(mysum)
```

Output:
```
21
```

```
mysum = 0
for i in range(7, 10):
    mysum += i
print(mysum)
```

Output:
```
24
```

For Examples

```
x = 4
for i in range(x):
    print(i)
```

Output:

```
0
1
2
3
```

```
x = 4
for i in range(0,x):
    print(i)
```

Output:

```
0
1
2
3
```

For Examples - Strings

```
for ch in 'abcdef':
    print(ch)
```

Output:

```
a
b
c
d
e
f
```

```
letters = 'abcdef'
for i in range(0, len(letters)):
    print(letters[i])
```

Output:

```
a
b
c
d
e
f
```

For Examples - Strings

The following code segments do the same thing

The second one is more “pythonic”

```
s = "abcdefgh"
for index in range(len(s)):
    if s[index] == 'i' or s[index] == 'u':
        print("There is an i or u")

for char in s:
    if char == 'i' or char == 'u':
        print("There is an i or u")
```

break STATEMENT

The break statement is used to immediately exit whatever loop it is in.

Break skips remaining expressions in code block.

Break exits only containing loop!

```
for <var> in <sequence>:
    <statement>
    break
    <statement>
```

`break` Statement Examples

```
mysum = 0
for i in range(5, 11, 2):
    mysum += i
    if mysum == 5:
        break
    mysum += 1
print(mysum)
```

For Loop Example

Write a Python program to input the status (F - Full-time, P-Part-time) and the salary of 10 instructors and output:

- the number of <u>full-time</u> instructors.
- the <u>average</u> salary of <u>all</u> instructors.

- See: `week3_salary.py`

Write a Python program to input 12 temperature values (one for each month) and display the number of the month with the highest temperature.

- See: `week3_temperatures.py`

Write a Python guessing game program. The program should generate a random int between 1 and 10. The user has 3 guesses to guess correctly. The program should output an appropriate message if the user guesses correctly/does not guess correctly.

- See: `week3_game.py`

for VS while LOOPS

`for` loops:

- know number of iterations
- can end early via break
- uses a counter
- can rewrite a for loop using a while loop

`while` loops

- unbounded number of iterations
- can end early via break
- can use a counter but must initialize before loop and increment it inside loop
- may not be able to rewrite a while loop using a for loop

Nested Loops

Nested Loop means a loop within another loop.

For each iteration of the outer loop is executed, the inner loop is executed completely.

```
for i in range(1,6,2):
    for j in range(6, 0, -3):
        print("i = " + str(i) + "\tj = " + str(j))
```

Output:

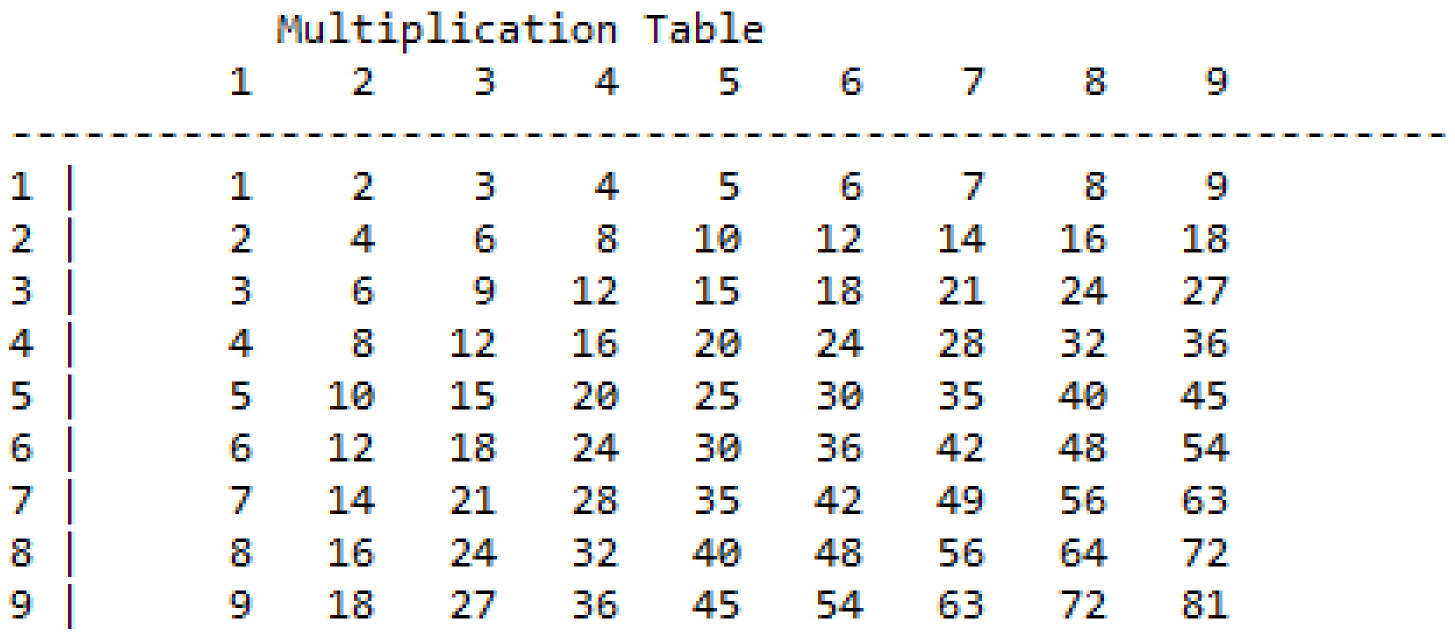
```
i = 1  j = 6
i = 1  j = 3
i = 3  j = 6
i = 3  j = 3
i = 5  j = 6
i = 5  j = 3
```

A program that prints a triangle of stars:

```
MAX_ROWS = 10;
for row in range(1, MAX_ROWS + 1,1):
    for star in range(1,row+1,1):
        print ("*", end=' ')
    print()
```

```
Output:
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
```

Nested for Loop



```
           Multiplication Table
         1    2    3    4    5    6    7    8    9
-----------------------------------------------------------
1 |      1    2    3    4    5    6    7    8    9
2 |      2    4    6    8   10   12   14   16   18
3 |      3    6    9   12   15   18   21   24   27
4 |      4    8   12   16   20   24   28   32   36
5 |      5   10   15   20   25   30   35   40   45
6 |      6   12   18   24   30   36   42   48   54
7 |      7   14   21   28   35   42   49   56   63
8 |      8   16   24   32   40   48   56   64   72
9 |      9   18   27   36   45   54   63   72   81
```

Nested for Loop - Stars

Write a program to display the triangle of stars below.

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

- See: 03_stars2.py

Exercise - What is Output?

```
s1 = "Bilkent u rock"
s2 = "i rule Bilkent"
if len(s1) == len(s2):
    for char1 in s1:
        for char2 in s2:
            if char1 == char2:
                print("common letter")
                break
```

Some Simple Numerical Programs

Exhaustive Enumeration (Guess and Check)

Exhaustive enumeration is a search technique that works only if the set of values being searched includes the answer.

Exhaustive Enumeration Example

```
#Find the cube root of a perfect cube

x = int(input('Enter an integer: '))
ans = 0
while ans **3 < abs(x):
    ans = ans + 1

if ans**3 != abs(x):
    print(x, 'is not a perfect cube')
else:
    if x < 0:
            ans = -ans
    print('Cube root of', x, 'is', ans)
```

Program Technique

In this program, we enumerate all possibilities until we find the right answer or exhaust the space of all possibilities.

This may not seem like an efficient way to solve a problem but exhaustive enumeration algorithms are often the most practical solution.

Such algorithms are easy to implement and understand.

In many cases, they run fast enough for all practical purposes.

GUESS-AND-CHECK – cube root

```
cube = 8
for guess in range(abs(cube)+1):
    if guess**3 >= abs(cube):
        break
if guess**3 != abs(cube):
    print(cube, 'is not a perfect cube')
else:
    if cube < 0:
        guess = -guess
    print('Cube root of '+str(cube)+' is '+str(guess))
```

APPROXIMATE SOLUTIONS

- **good enough** solution
- start with a guess and increment by some **small value**
- keep guessing if `|guess`^{3}`-cube| >= epsilon` for some **small epsilon**

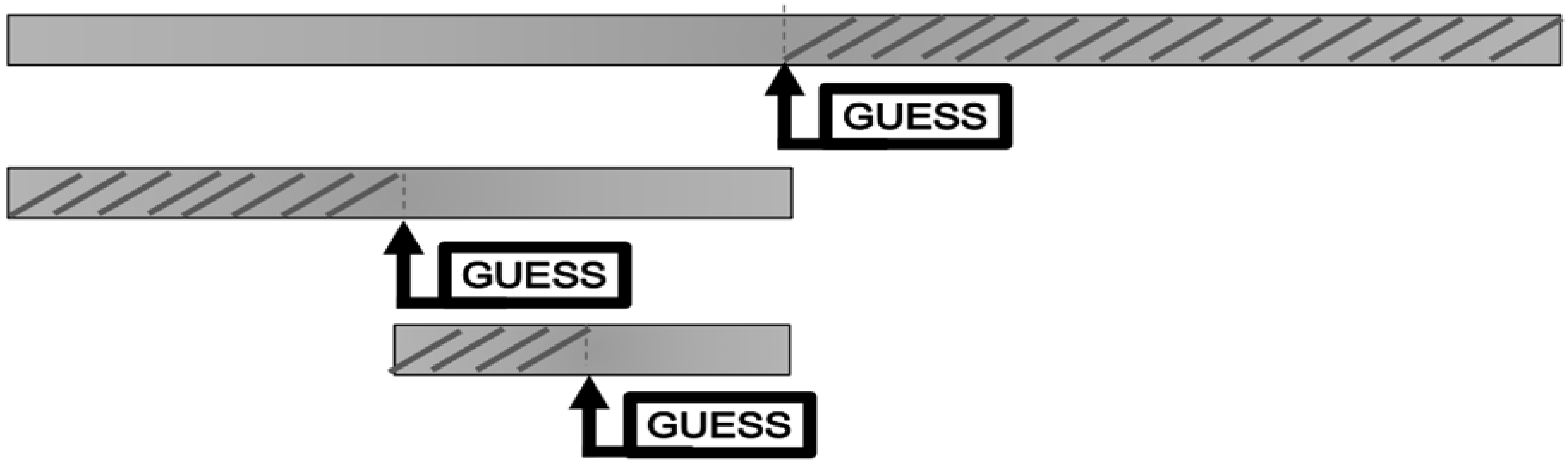
- decreasing increment size → slower program
- increasing epsilon → less accurate answer

APPROXIMATE SOLUTION – cube root

```
cube = 27
epsilon = 0.01
guess = 0.0
increment = 0.0001
num_guesses = 0
while abs(guess**3 - cube) >= epsilon  and guess <= cube :
    guess += increment
    num_guesses += 1
print('num_guesses =', num_guesses)
if abs(guess**3 - cube) >= epsilon:
    print('Failed on cube root of', cube)
else:
    print(guess, 'is close to the cube root of', cube)
```

BISECTION SEARCH

- half interval each iteration
- new guess is halfway in between
- to illustrate, let's play a game!



BISECTION SEARCH – cube root

```
cube = 27
epsilon = 0.01
num_guesses = 0
low = 0
high = cube
guess = (high + low)/2.0
while abs(guess**3 - cube) >= epsilon:
    if guess**3 < cube :
        low = guess
    else:
        high = guess
    guess = (high + low)/2.0
    num_guesses += 1
print('num_guesses =', num_guesses )
print(guess, 'is close to the cube root of', cube)
```

Bisection Search - Program Technique

At each iteration of the loop in the program, the size of the space to be searched is cut in half.

Because it divides the search space in half at each step, it is called a **bisection search**.

Bisection search is a huge improvement over exhaustive enumeration because exhaustive enumeration reduced the search space by only a small amount at each iteration.

Terms of Use