

# Syllabus

* Control Structures
* Repetitive Structures

# Structured Programming

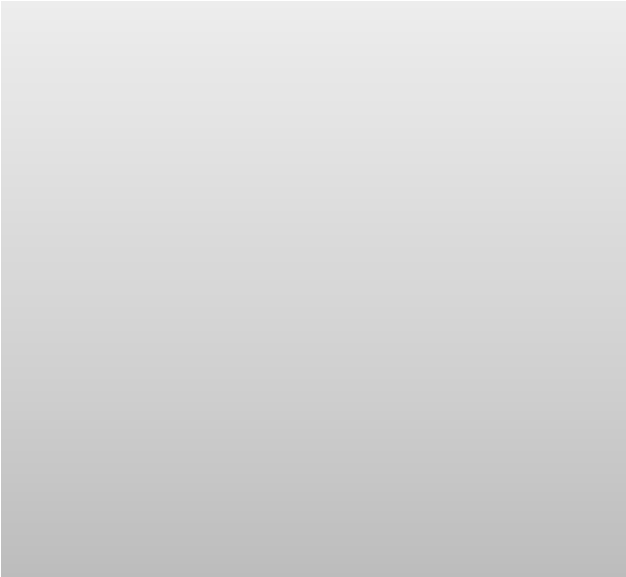
* Any computing problem can be solved by executing a series of actions in a certain specific order – this is called structured programming
* An algorithm is the sequence of steps to be followed to solve a problem
* Specifying the order in which the statements should execute in a computer program is called program control
* Structured programming promotes simplicity in problem solving approach
* Bohm and Jacopini have given us the result that only three forms of control are needed:
  + Sequence
  + Selection
  + Repetition

# Control Structures

* Statements in the program are executed in the order in which they are written – this is called **sequential execution**
* Certain statement can decide which should be the next statement to be executed – this is called **transfer of control**
* Transfer of control is generally based on a condition checking
* Python provides the following structures for transfer of control:
  + **if** for single selection
  + **if/else** for double selection
  + **if/elif/else** for multiple selection

# Truth Values and Boolean Tests

* All objects have an inherent Boolean true or false value.
* Any nonzero number or nonempty object is true.
* Zero numbers, empty objects, and the special object **None** are considered false.
* Comparisons and equality tests are applied recursively to data structures.
* Comparisons and equality tests return True or False (custom versions of 1 and 0).
* Boolean **and** and **or** operators return a true or false operand object.
* Boolean operators stop evaluating (“short circuit”) as soon as a result is known.



>>>

a =

0

>>>

b =

0

>>>

c =

5

>>>

a and b

0

>>>

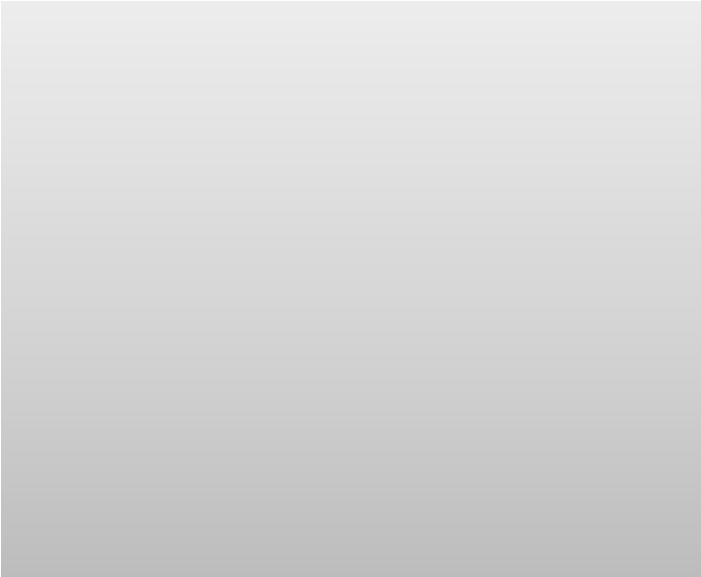
c and a

0

>>>

c or a

5



>>>

b =

9

>>>

b or c

9

>>>

c or b

5

>>>

not c

False

>>>

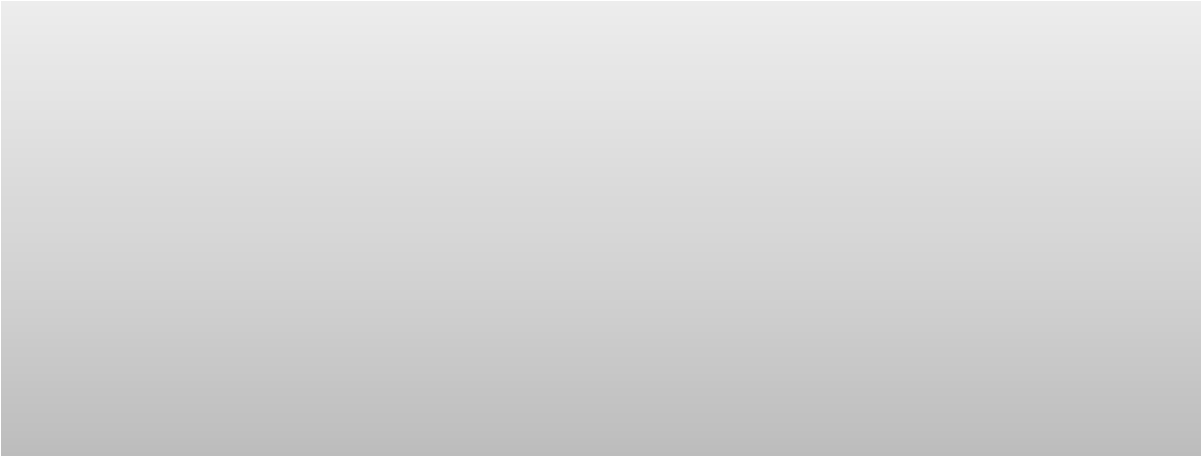
not a

True



Short-circuit behavior

Interesting Use of Truth Values



0

>>>

A =

0

B =

>>>

4

C =

>>>

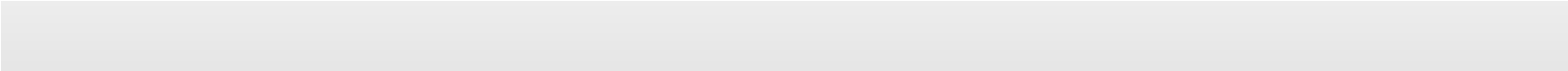
X = A or B or C or None

>>>

X

>>>

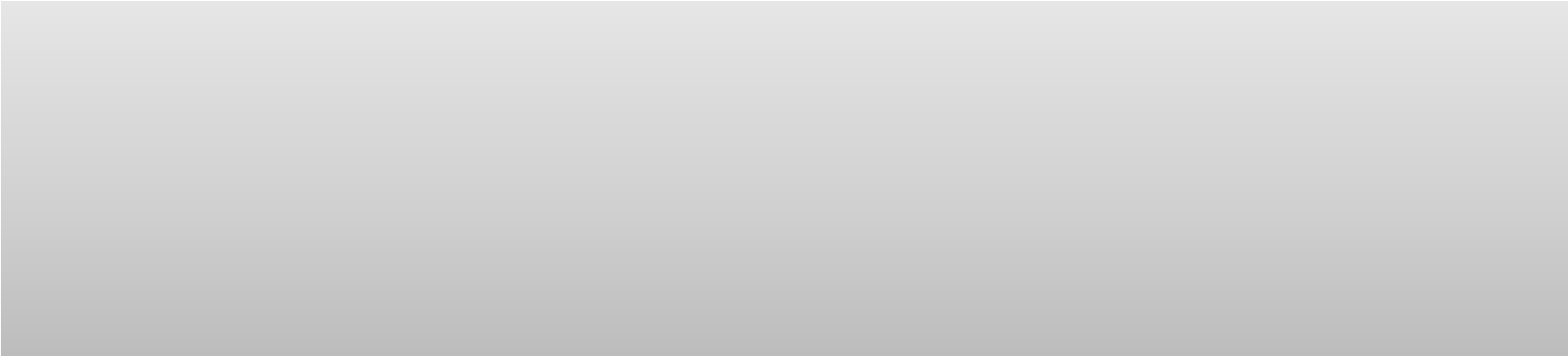
4



L = [1, 0, 2, 0, 'Spaniel', '', 'Shepherd',

>>>

[]]



))

list(filter(bool, L

>>>

[1

]

, 2, 'Spaniel', 'Shepherd'

]

x for x in L if x

>>> [

]

, 2, 'Spaniel', 'Shepherd'

[1

)

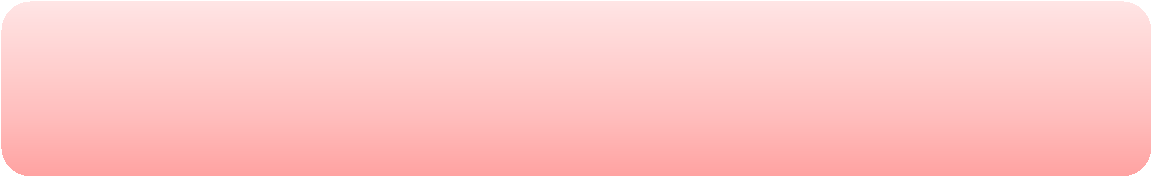
any(L), all(L

>>>

)

True, False

(



The

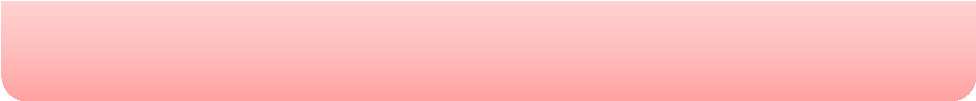
**any**

and

**all**

built-ins can be used to test if any or

all items in a collection are true



**bool**

function will translate X into the

equivalent of integer 1 or 0



Comprehension

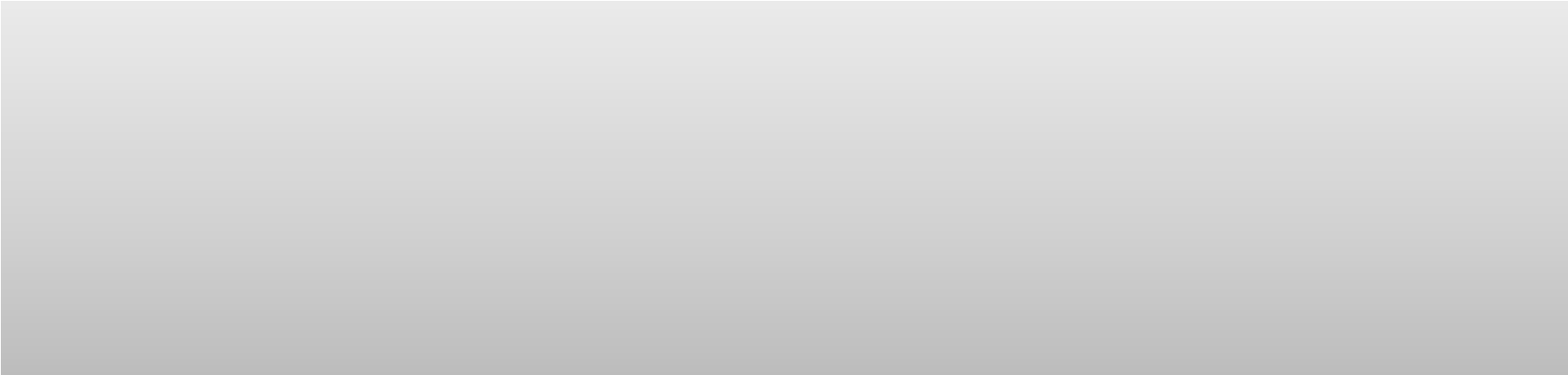
# Relational Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| > | Greater than |
| >= | Greater than or equal to |
| < | Less that |
| <= | Less than or equal to |
| == | Equal to |
| != | Not equal to |

# Control Structure: **if**

* A single entry, single exit structure
* Execute the statement if the given condition is TRUE, otherwise the statement or the block of statements

is skipped



>>>

age = input('Enter your age: '

)

Enter your age: 18

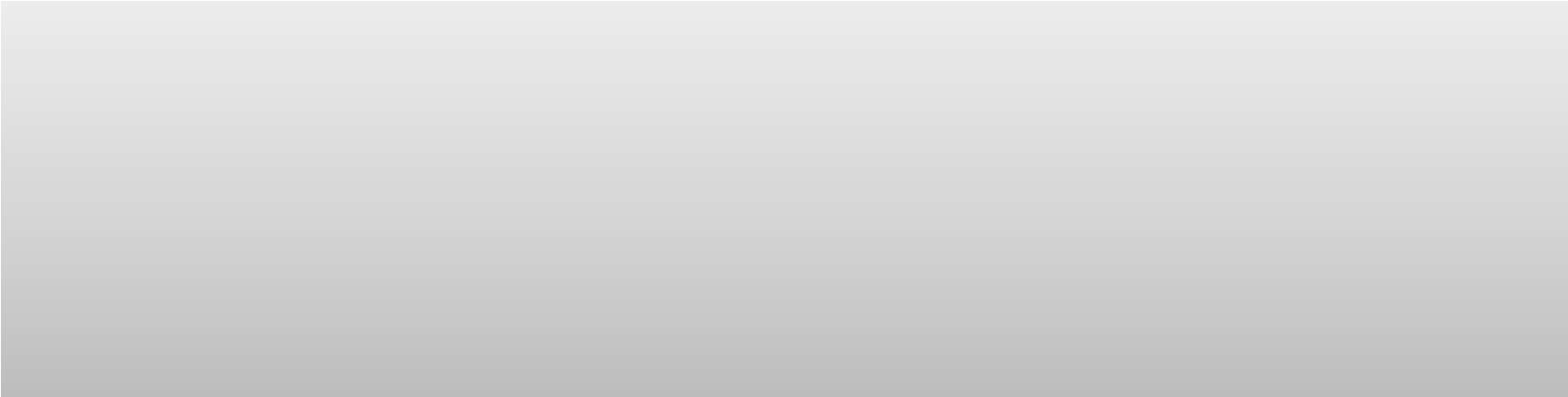
>>>

if int(age)

>= 18:

print('You can see movie: \'The Revenant\'')

You can see movie: 'The Revenant'



>>>

age = input('Enter your age: '

)

Enter your age: 14

>>>

if int(age)

>= 18:

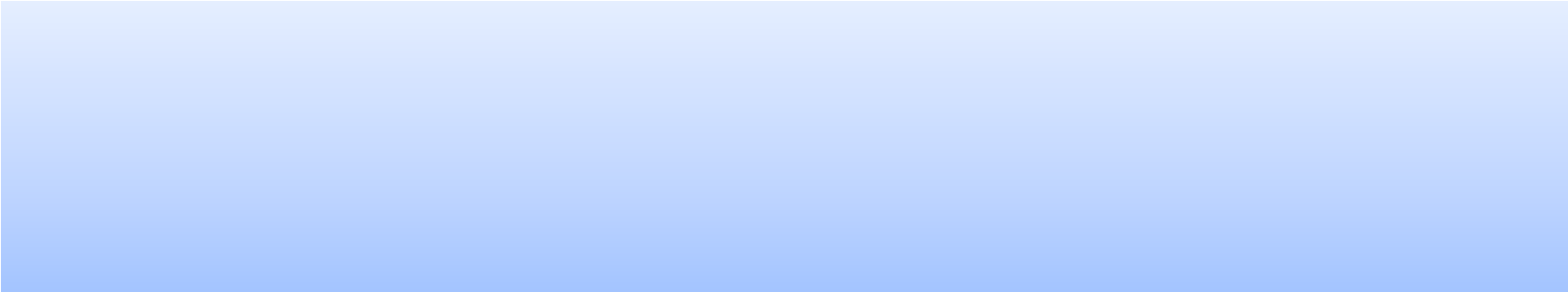
print('You can see movie: \'The Revenant\'')

>>>

# Control Structure: **if/else**

• In this construct, first the condition in the **if** clause is evaluated

* If it is TRUE then the statements below **if** is executed
* Otherwise the statements below else will be executed



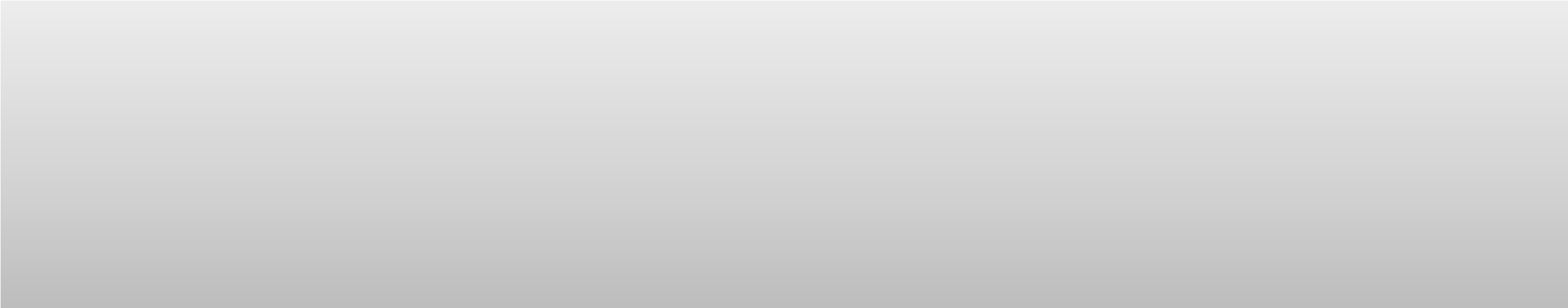
age = input('Enter your age: ')

if int(age) >= 18:

print('You can see movie: \'The Revenant\'')

else:

print('You can see movie: \'KungfuPanda\'')



>>>

Enter your age: 55

You can see movie: 'The Revenant'

>>> ================================ RESTART ================================

>>>

Enter your age: 14

You can see movie: 'KungfuPanda'

>>>

# A Practical Example: **if/else**

• Write a program to accept a string from the user and check whether the string is a palindrome or not

# Solution



# This program check if the user input is a

# palindrome or not

input\_string= input('Enter the string: ')

# Reverse the string

rev\_string= input\_string[::-1]



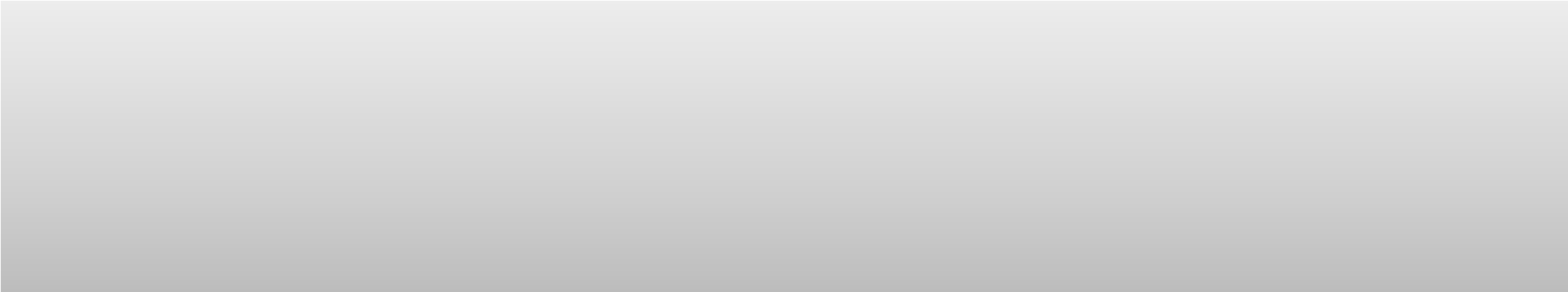
# Compare the strings

if input\_string== rev\_string:

print('The entered string is a palidrome')

else:

print('The entered string is not a plaindrome')



>>>

Enter the string: malayalam

The entered string is a palidrome

>>> ================================ RESTART ================================

>>>

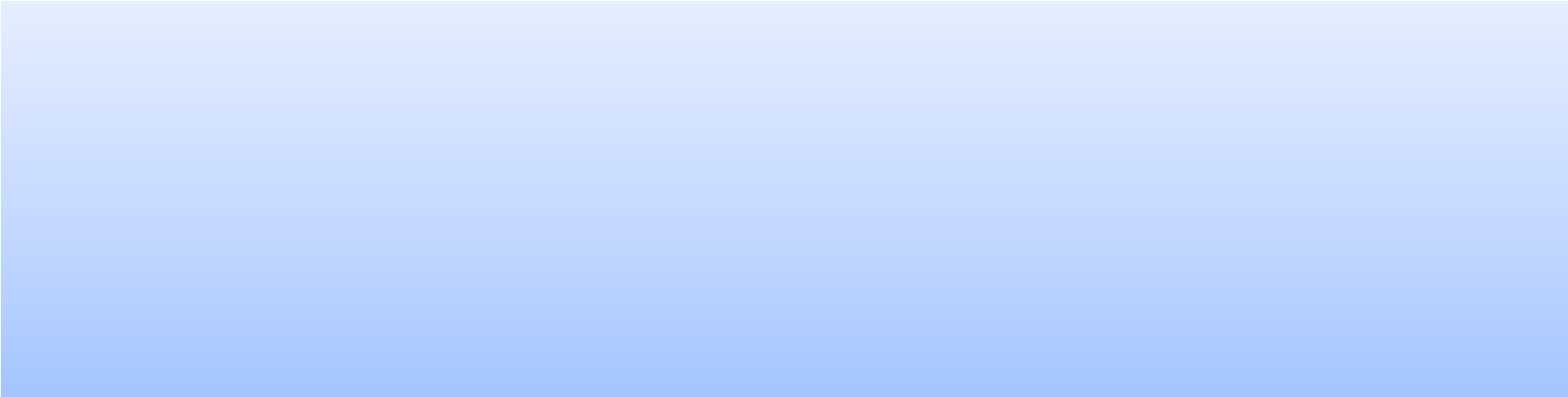
Enter the string: palindrome

The entered string is not a plaindrome

# Control Structure: **if/elif/else**

* In this control structure, conditions are evaluated from the beginning of the structure
* At the first condition that is evaluated to be TRUE
  + The statement or the block of statements are executed
  + The control exits from the structure even if other condition could be TRUE
* This construct is very useful in taking priority based actions

# Control Structure: **if/elif/else**



age = input('Enter your age: ')

if int(age) >= 18:

print('You can see movie: \'The Revenant\'')

elif18 > int(age) > 13:

print('You can see movie: \'KungfuPanda\'')

else:

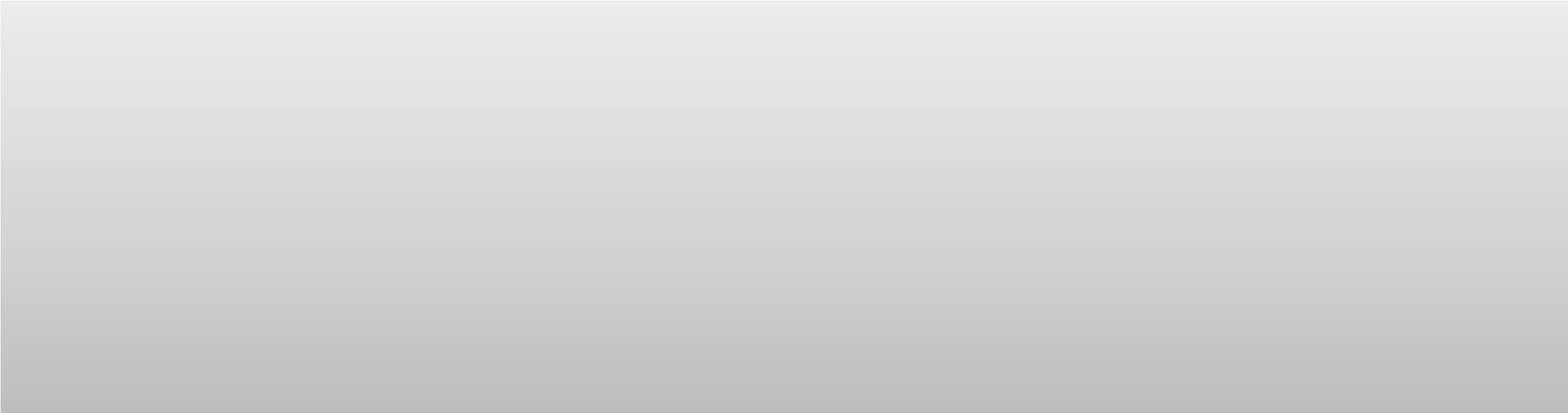
print('Better watch Tom and Jerry');



You can have as many

**elif**

conditions as you want



>>>

Enter your age: 12

Better watch Tom and Jerry

>>> ================================ RESTART ================================

>>>

Enter your age: 15

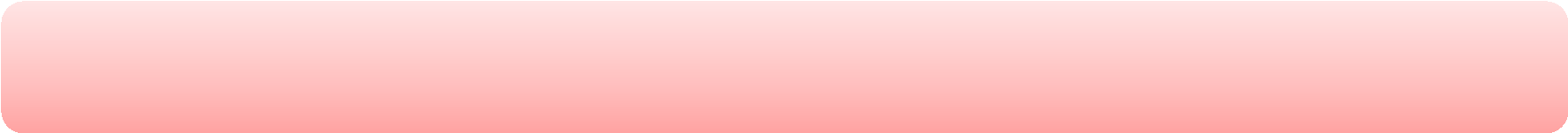
You can see movie: 'KungfuPanda'

>>> ================================ RESTART ================================

>>>

Enter your age: 18

You can see movie: 'The Revenant'

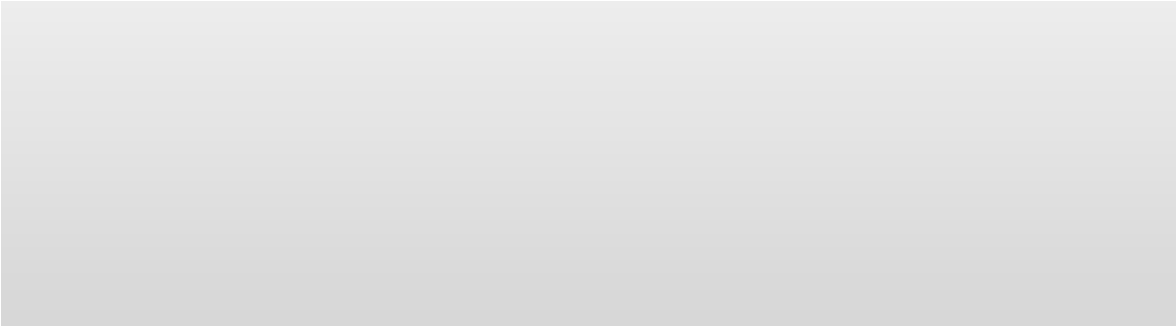


Testing every condition is mandatory for all script

s small or big

# **if/else** Ternary Expression

• A single line version of **if/else** construct



>>>

a =

43

>>>

b =

76

>>>

c =

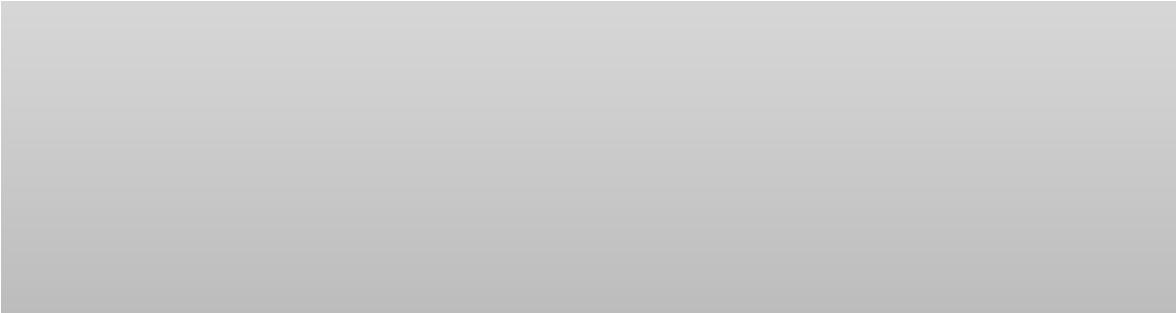
0

>>>

y = a ifc elseb

>>>

y



76

>>>

c =

1

>>>

y = a ifc elseb

>>>

y

43

# Nested **if** Structures

* It is often necessary to include an if statement inside another. Such an arrangement is called nested set of decision statements
* Nested decisions are required for problems that have multiple levels of decision making

# A Practical Example: **if**

• Now compute the taxes due, given a marital status and an income figure.

|  |  |  |
| --- | --- | --- |
| **Federal Tax Schedule** | |  |
| If your status is single and your taxable income is | Tax is | Of the amount over |
| At most $32000 | 10% | $0 |
| More than $32000 | $3200 + 25% | $32000 |
| If your status is married and your taxable income is | Tax is | Of the amount over |
| At most $64000 | 10% | $0 |
| More than $64000 | $6400 + 25% | $64000 |

The key point is that there are two levels of decision making. First, you must branch on the marital status. Then, for each marital status, you must have another branch on income level.



# Solution



# Initialize constant variables for the tax rates and rate limits

RATE1 = 0.10

RATE2 = 0.25

RATE1\_SINGLE\_LIMIT = 32000.0

RATE1\_MARRIED\_LIMIT = 64000.0

# Read income and marital status

income = float(input("Please enter your income: "))

maritalStatus= input("Please enter s for single, m for married: ")

# Compute taxes due

tax1 = 0



tax2 = 0

if maritalStatus== "s" :

if income <= RATE1\_SINGLE\_LIMIT :

tax1 = RATE1 \* income

else :

tax1 = RATE1 \* RATE1\_SINGLE\_LIMIT

tax2 = RATE2 \* (income -RATE1\_SINGLE\_LIMIT)

else :

if income <= RATE1\_MARRIED\_LIMIT :

tax1 = RATE1 \* income

else :

tax1 = RATE1 \* RATE1\_MARRIED\_LIMIT

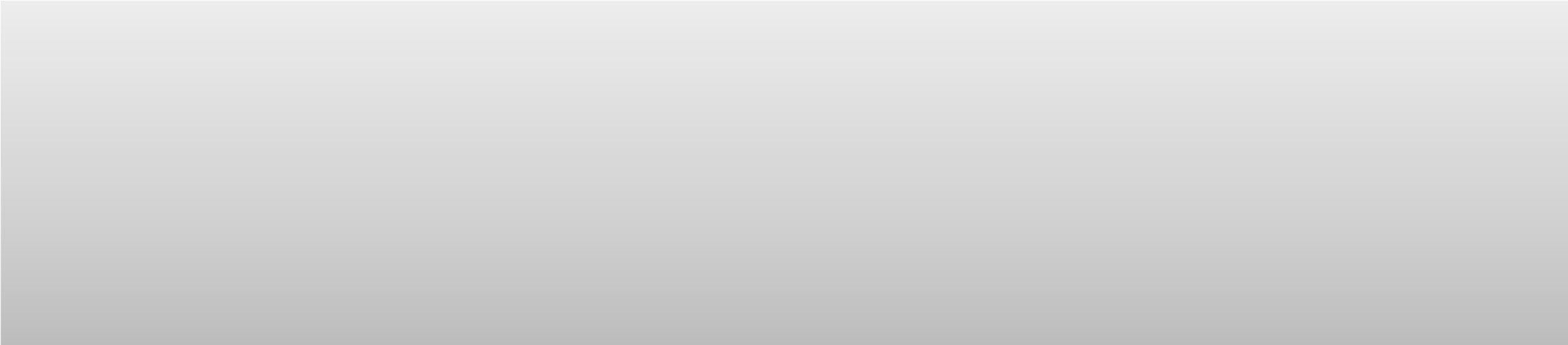
tax2 = RATE2 \* (income -RATE1\_MARRIED\_LIMIT)

totalTax= tax1 + tax2

# Print the results

print("The tax is $%.2f" % totalTax)

# Output



>>>

Please enter your income: 36000

Please enter s for single, m for married: s

The tax is $4200.00

>>> ================================ RESTART ================================

>>>

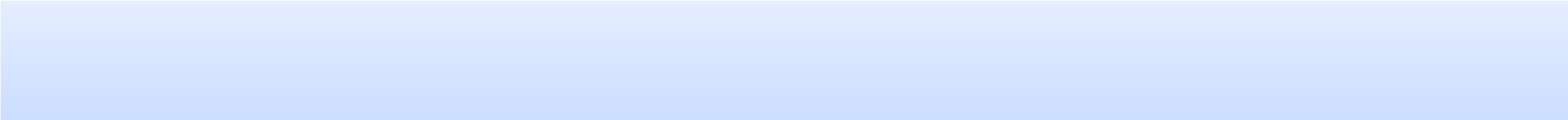
Please enter your income: 98000

Please enter s for single, m for married: m

The tax is $14900.00

# Indentation Errors

• Improper indentation can cause syntax errors as shown in the below example:



x = 'SPAM'

**# Error: first line indented**

if 'rubbery' in 'shrubbery':

print(x \* 8)



x += 'NI'

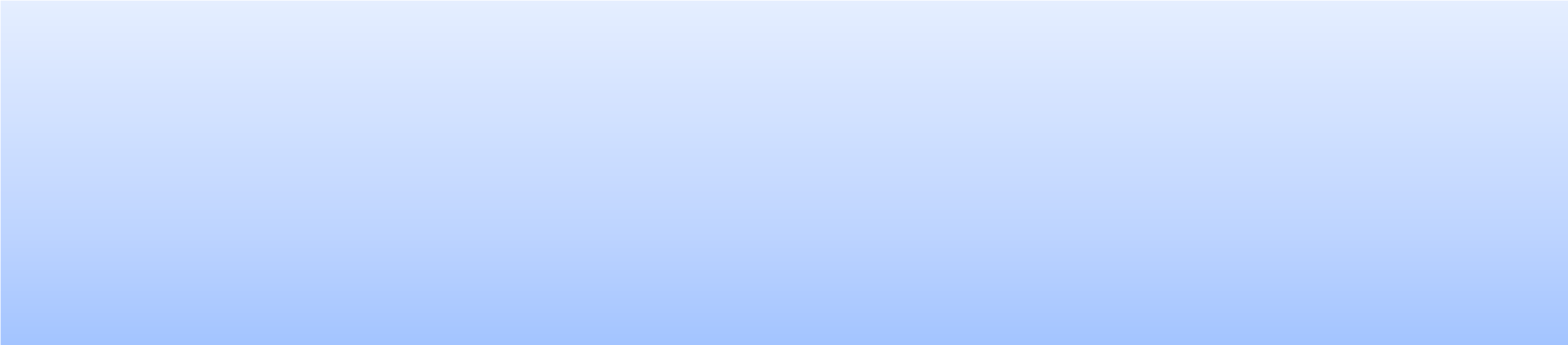
**# Error: unexpected indentation**

if x.endswith('NI'):

x \*= 2

print(x)

**# Error: inconsistent indentation**



**# Corrected version**

x = 'SPAM'

if 'rubbery' in 'shrubbery':

print(x \* 8)

**# Prints8 "SPAM"**

x += 'NI'

if x.endswith('NI'):

x \*= 2

print(x)

**# Prints "SPAMNISPAMNI"**

# Repetition Structure

* A repetition structure allows to run a statement or a block of code to repeat itself until a specified condition is met
* Also called looping structures
* Python provides two such structures:
  + **for**
  + **while**

# Repetition Structure: **for**

* The for loop is a generic iterator in Python: it can step through the items in any ordered sequence or other iterable object
* The for statement works on strings, lists, tuples and other built-in iterables, as well as new user-defined objects
* General format:

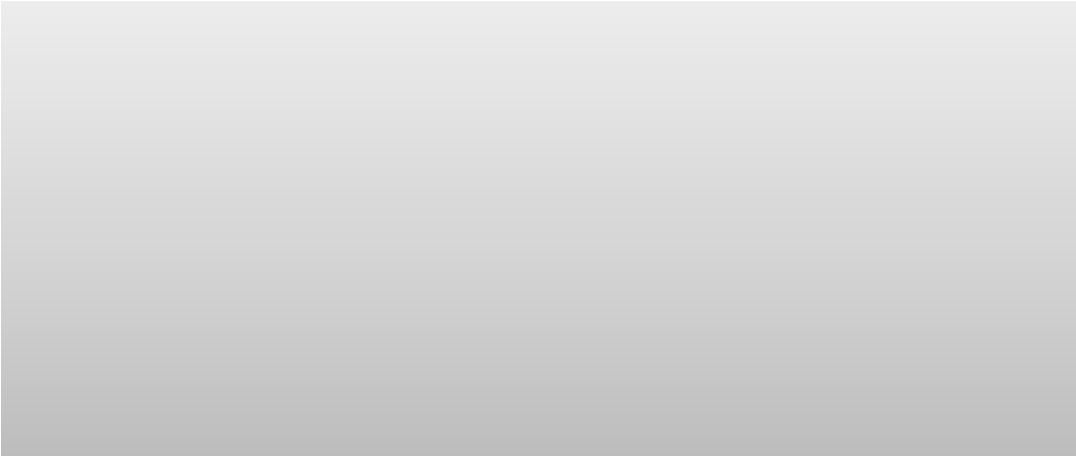
for target in object: # Assign object items to target statements # Repeated loop body: use target



else: # Optional else part

statements **# If we didn't hit a 'break**'

* Example:



>>>

sum =

0

5]:

for item in [1, 2, 3, 4,

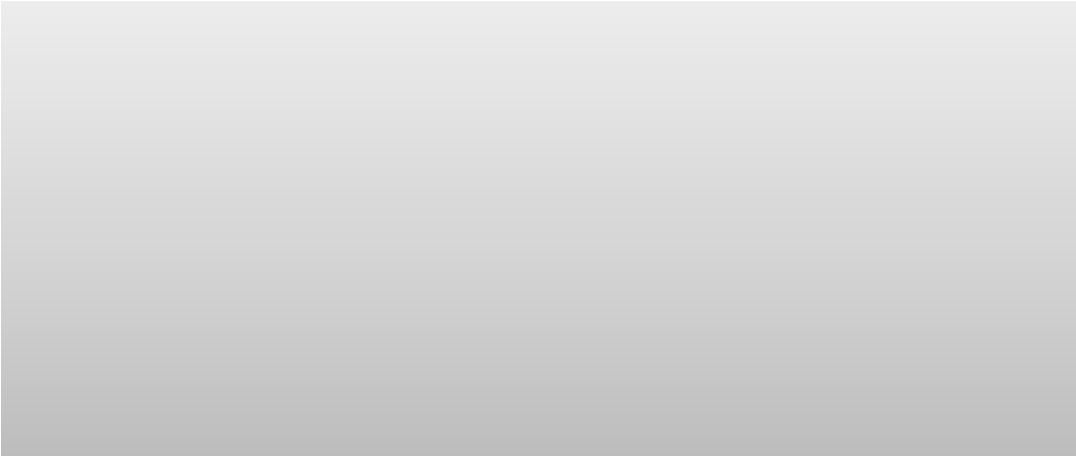
>>>

sum += item

sum

>>>

15



6)]

T = [(1, 2), (3, 4), (5,

>>>

:

>>>

for (x, y) in T

print(x, y)

1

2

3

4

5

6

# **range()**

• The function generates a list of numbers, which is generally used to iterate over with for loops • It has the syntax: **range([start], stop[, step])**

* start: Starting number of the sequence.
* stop: Generate numbers up to, but not including this number.
* step: Difference between each number in the sequence.

## A Practical Example: **for**

• Develop a python script to generate the first N Fibonacci numbers

## Solution



# Program to generate first N Fibonacci numbers

# Get the user input N

n = int(input('Enter the number of Fibonacci number to generate: '))

# Initialize the values

a = 0

b = 1



# Loop through to produce the numbers

print (a)

print (b)

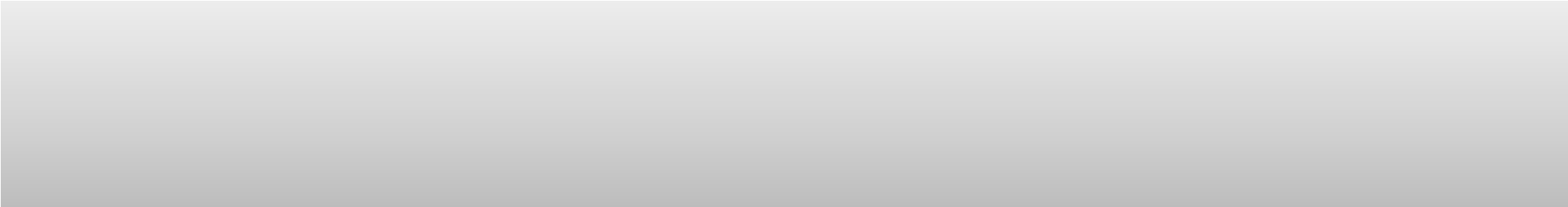
for iin range(n-2):

x = a + b;

print(x)

a = b

b = x



Enter the number of Fibonacci number to generate: 10

0

1

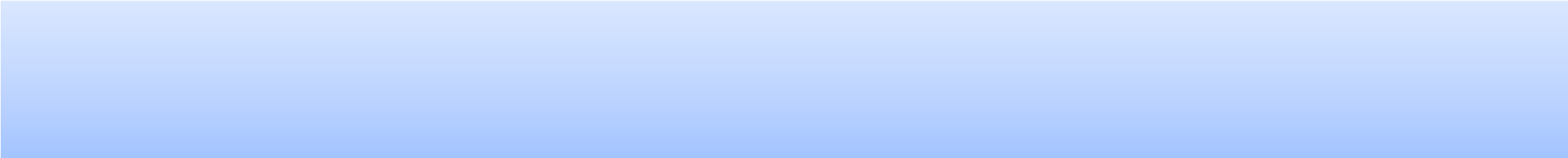
1

2 3 5 8 13 21

34

## Repetition Structure: **while**

* A while structure continuously repeats a block of statements until the condition specified at the top keeps evaluating to TRUE
* General format:



while condition:

**# Loop test**

statements

**# Loop body**

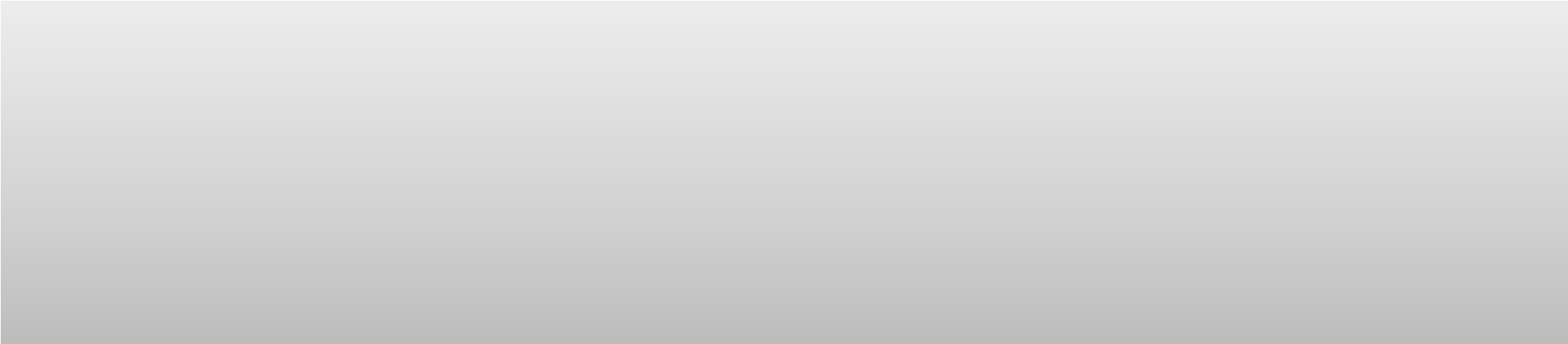
else:

**# Optional else**

statements

**# Run if didn't exit loop with break**

* Example:



>>>

S = 'BillieJean'

>>>

while S

:

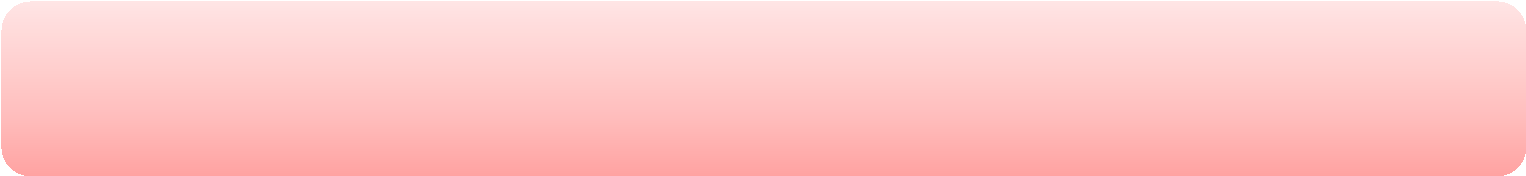
print(S, end=' ')

S = S[1:]

BillieJeanillieJeanllieJeanlieJeanieJeaneJeanJean eanan n

## A Practical Example: **while**

• You put $10,000 into a bank account that earns 5 percent interest per year. How many years does it take for the account balance to be double the original investment?



Develop an algorithm for this

## Solution



# Get inputs from the user

RATE = float(input('Enter Rate of Interest: '))

INITIAL\_BALANCE = float(input('Enter Principle Amount: '))

TARGET = 2 \* INITIAL\_BALANCE

# Initialize variables used with the loop

balance = INITIAL\_BALANCE

year = 0

# Count the years required for the investment to double

while balance < TARGET :



year = year + 1

interest = balance \* RATE / 100

balance = balance + interest

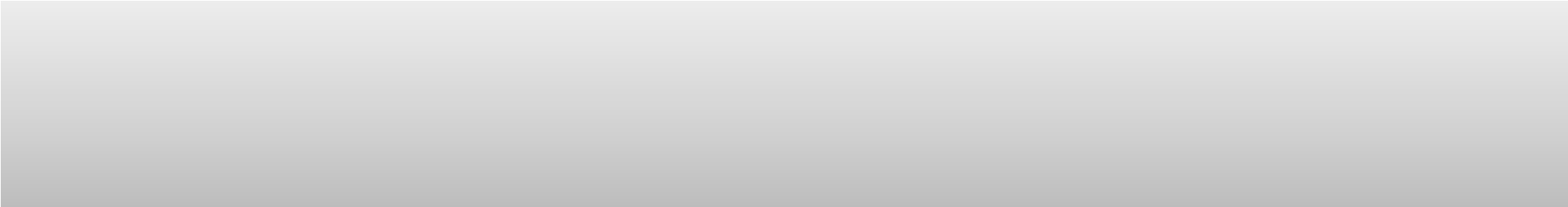
# Print the results

print("The investment doubled after", year, "years.")



Trace this program by hand

## Output



>>>

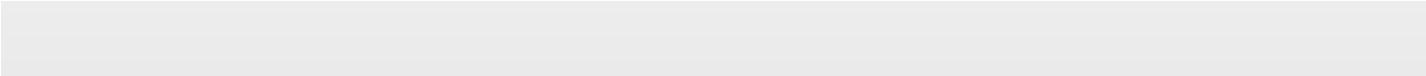
Enter Rate of Interest: 5.0

Enter Principle Amount: 10000.00

The investment doubled after 15 years.

## **break** Statement

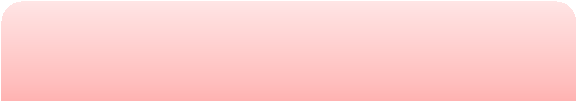
* The break statement causes an immediate exit from a loop
* Because the code that follows it in the loop is not executed if the break is reached, you can also sometimes avoid nesting by including a break



:

while True

>>>



name = input('Enter name: ')

if name == 'stop': break

print('Hi!' + name)

Enter name: Christopher Nolan

Hi!ChristopherNolan

Enter name: MichealBay

Hi!MichealBay

Enter name: Steven Spielberg

Hi!StevenSpielberg

Enter name: George Lucas

Hi!GeorgeLucas

Enter name: Stop

Hi!Stop

Enter name: stop

\



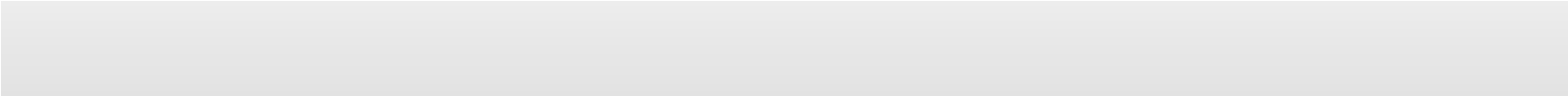
Sentinel Value



Infinite Loop

## **continue** Statement

• The continue statement causes an immediate jump to the top of a loop



>>>

for n in range

(25):



if n % 2 == 0: continue

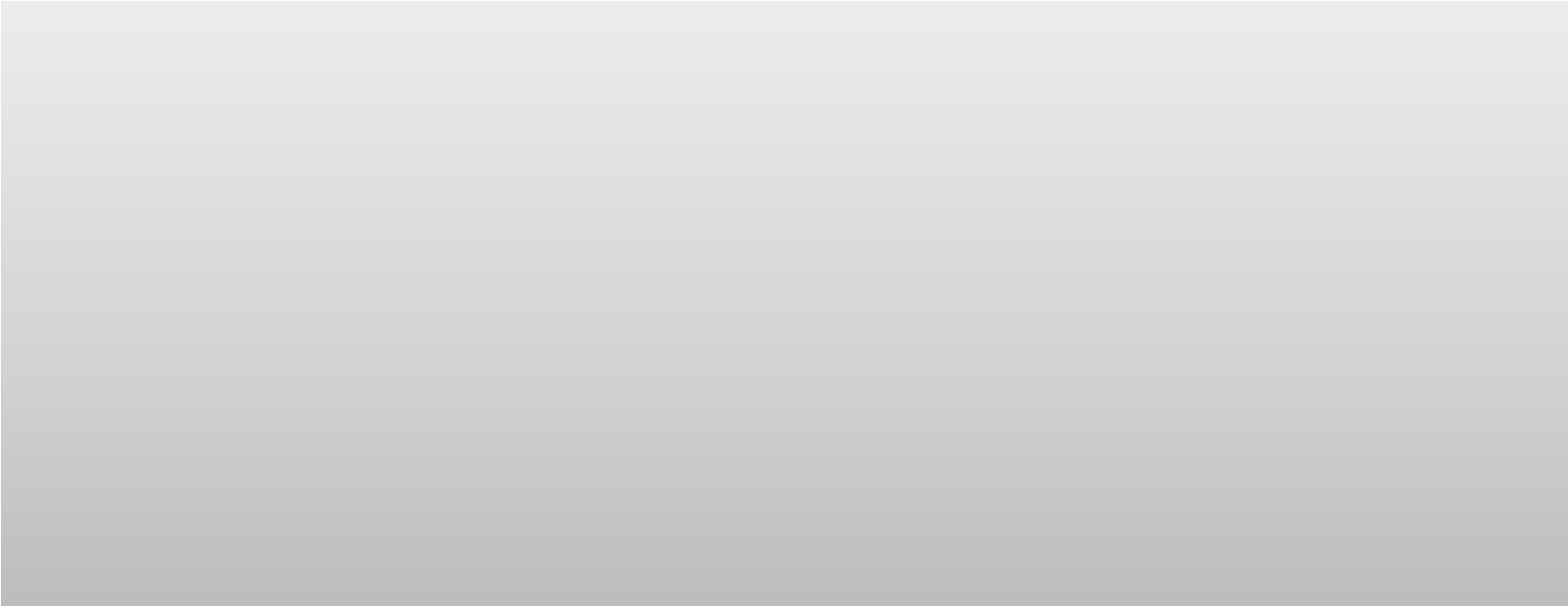
print(n, end=' ')

1

3 5 7 9 11 13 15 17 19 21 23

## **pass** Statement

* **pass** statement is a no-operation placeholder that is used when the syntax requires a statement, but you have nothing useful to say
* It is often used to code an empty body for a compound statement



>>>

for n in range

(15):

print(n, end=' ')

if n%5 != 0: pass

else: print('Divisible by 5')

0

Divisible by

5

1

2 3 4 5 Divisible by

5

6

7 8 9 10 Divisible by

5

11

12 13

14

## The Loop **else** Block

• This block executes when the control exits the loop without encountering the break statement



# Program to check if a number is prime

y = int(input('Enter the number: '))

x = y//2

while x > 1:

if y%x== 0:



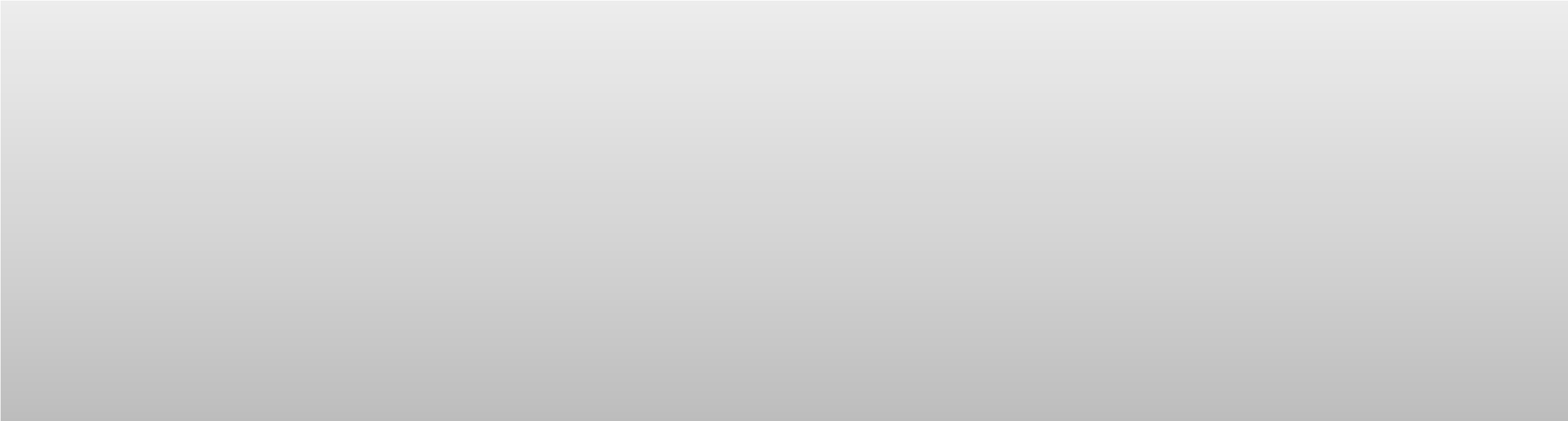
print(y, ' has factor ',x)

break

x -= 1

else: # This is a normal exit

print(y, ' is a prime number')



>>>

Enter the number: 5

5

is a prime number

>>> ================================ RESTART ================================

>>>

Enter the number: 67

67

is a prime number

>>> ================================ RESTART ================================

>>>

Enter the number: 45

45

has factor

15

## Nested Loop Structures

* Like the nested control statements, some complex iterations need a nested loop
* When the body of the loop consists of another loop, the loops are nested

A Practical Example: Nested Loops

• Write a script to print a table of powers of x

## Solution



# This program prints a table of powers of x.

# Initialize constant variables for the max ranges.

NMAX = 4

XMAX = 10

# Print table header.

for n in range(1, NMAX + 1) :

print("%10d" % n, end="")



print()

for n in range(1, NMAX + 1) :

print("%10s" % "x ", end="")

print("\n", " ", "-" \* 35)

# Print table body.

for x in range(1, XMAX + 1) :

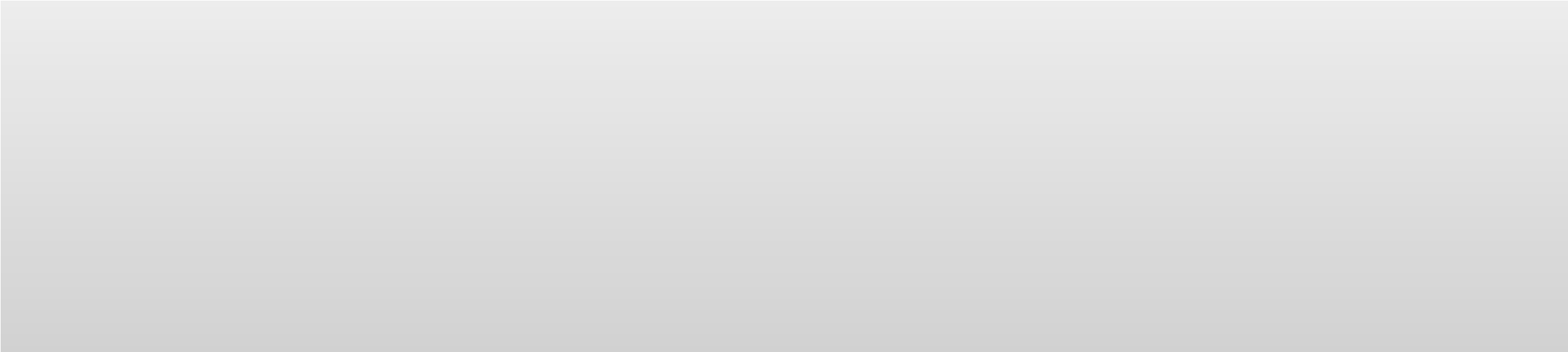
# Print the x row in the table.

for n in range(1, NMAX + 1) :

print("%10.0f" % x \*\* n, end="")

print()

## Output



>>>

1

2 3

4

x x

x

x

-----------------------------------

1

1

1 1

2

4 8

16

81

3

9 27

4

16 64

256

5

25 125

625



6

36 216

1296

7

49 343

2401

8

64 512

4096

9

81 729

6561

10

100 1000

10000

## Common Errors

* Inappropriate indentation can lead to syntax and logic errors
* Because floating-point values may be approximate, controlling the counting of loops with floating-point variables may result in imprecise counter values and inaccurate tests for termination. Programs should control counting loops with integer values
* Forgetting that the last value of the sequence returned by function range is one less than the value of the function’s end argument can lead to an **off-by-one** logic error
* Creating a for structure that contains no body statements is a syntax error.

## A Practical Application

* Application: Coordinate Geometry
* Write a program to check if a given set of 3 points A(x1,y1), B(x2,y2) and C(x3,y3) form a right angled isosceles triangle or not.
* Coordinate values should be taken from the user

Follow problem solving steps starting from information gathering



Time: 20 mins + 10 mins

## Solution



# Program to check if a given set of three points

# ax, ay, bx, by and cx, xyform a right angled isocelestriangle or not

import math

isosceles = False

right\_angled= False

# Input all the co-ordinates



print('Enter the coordinates of A: ')

ax = int(input('AX = '))

ay = int(input('AY = '))

print('Enter the coordinates of B: ')

bx= int(input('BX = '))

by = int(input('BY = '))

print('Enter the coordinates of C: ')

cx= int(input('CX = '))

cy = int(input('CY = '))

# Calculate the straight lines AB, BC and CA

ab= math.sqrt( (bx-ax)\*\*2 + (by -ay)\*\*2 )

bc= math.sqrt( (cx-bx)\*\*2 + (cy -by)\*\*2 )

ac = math.sqrt( (cx-ax)\*\*2 + (cy -ay)\*\*2 )

## Solution



# Check if any two sides are equal -isosceles condition

# Check with Pythogarastheorem -right angled condition

if ab== bc:

isosceles = True

if ((ab\*\*2) + (bc\*\*2)) == (ac\*\*2):

right\_angled= True

elifab== ac:

isosceles = True

if ((ab\*\*2) + (ac\*\*2)) == (bc\*\*2):



right\_angled

=

True

elifbc== ac:

isosceles = True

if ((bc\*\*2) + (ac\*\*2)) == (ab\*\*2):

right\_angled= True

else:

isosceles = False

right\_angled= False

# Print the result

if isosceles == True:

print('The points form an Isosceles triangle')

else:

print('The points do not form an Isosceles triangle')

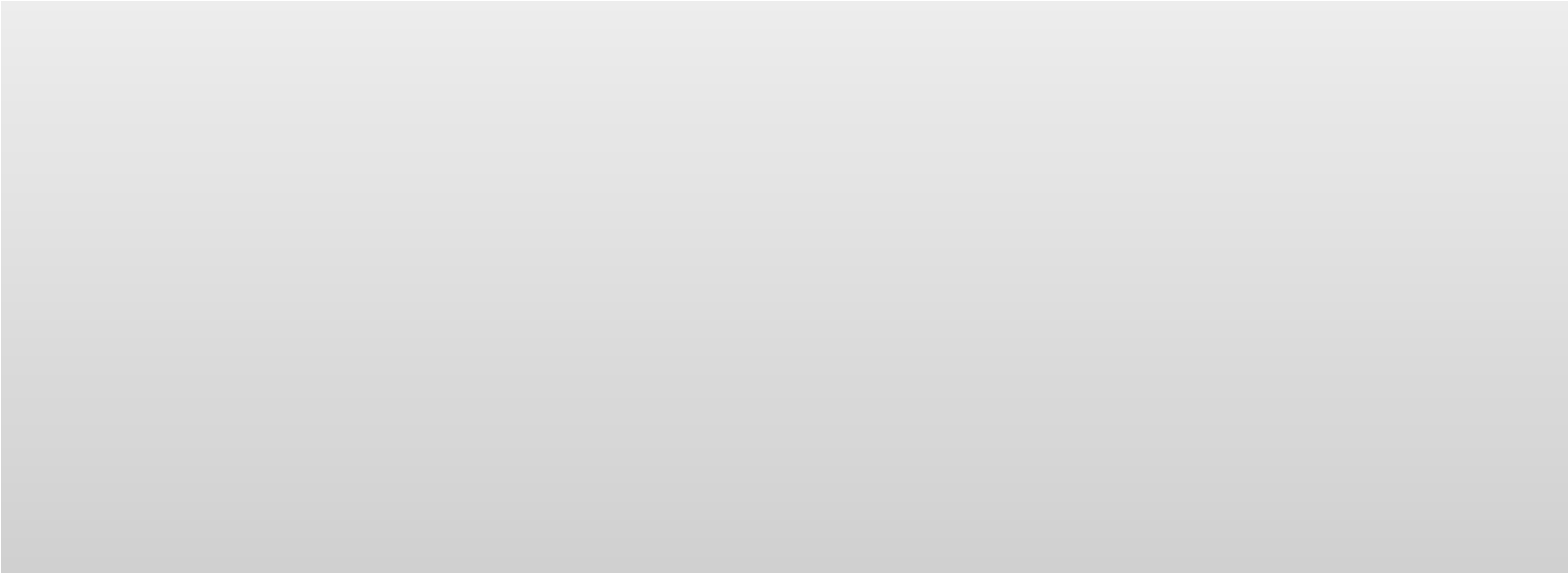
if right\_angled== True:

print('The points form an Right Angled triangle')

else:

print('The points do not form an Right Angled triangle')

## Output



Enter the coordinates of A:

AX = 3

AY = 0

Enter the coordinates of B:

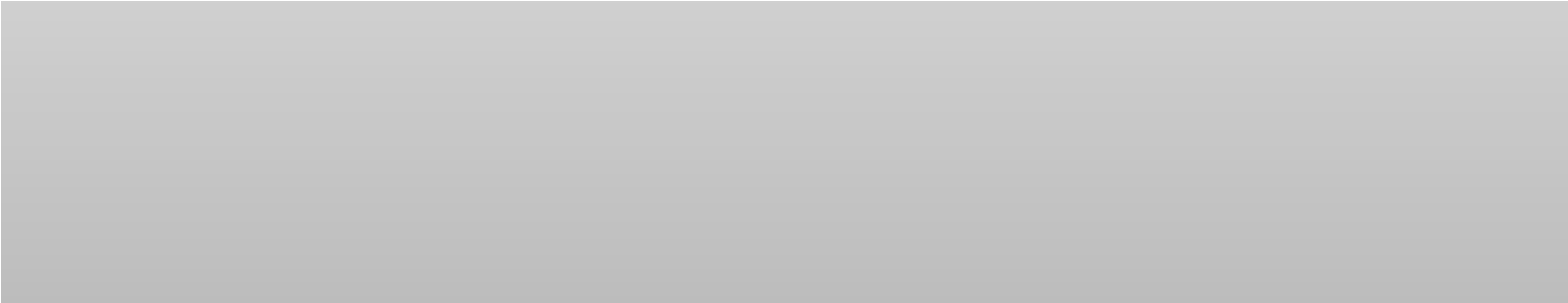
BX = 6

BY = 4

Enter the coordinates of C:

CX = -1

CY = 3



The straight lines have the dimensions:

3

0 6 4 -1

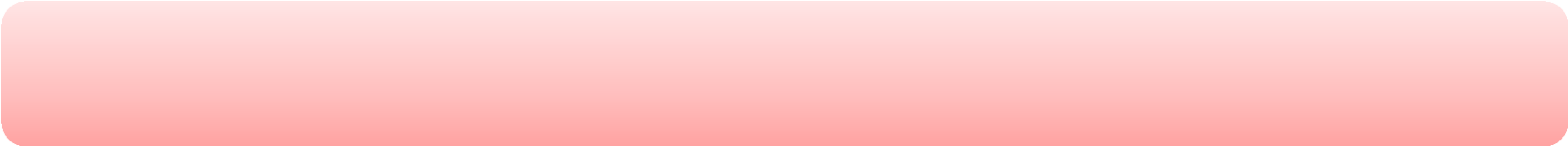
3

5.0 5.0 7.07106781187

The points form an Isosceles triangle

The points do not form a Right Angled triangle

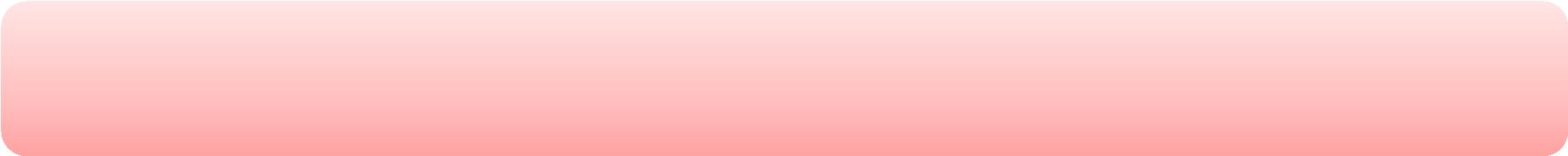
The chosen test case actually forms a right angled isosceles triangle. But we are getting unexpected result. Debug why? Investigate…



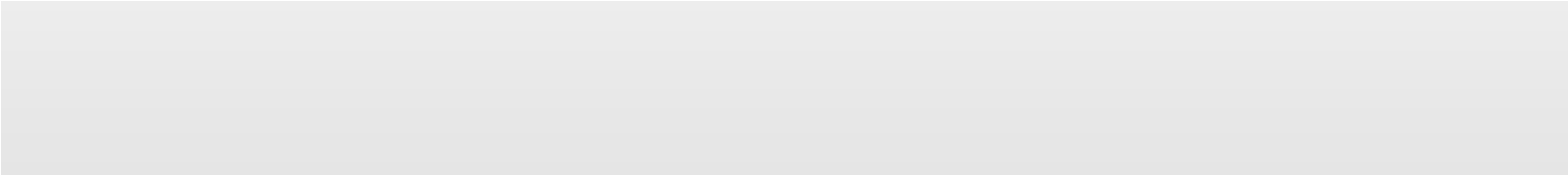
Refer: ***right\_angled\_isoceles.py*** given to you

What should be fixed?

After fixing some parts of the code, we get the following result.

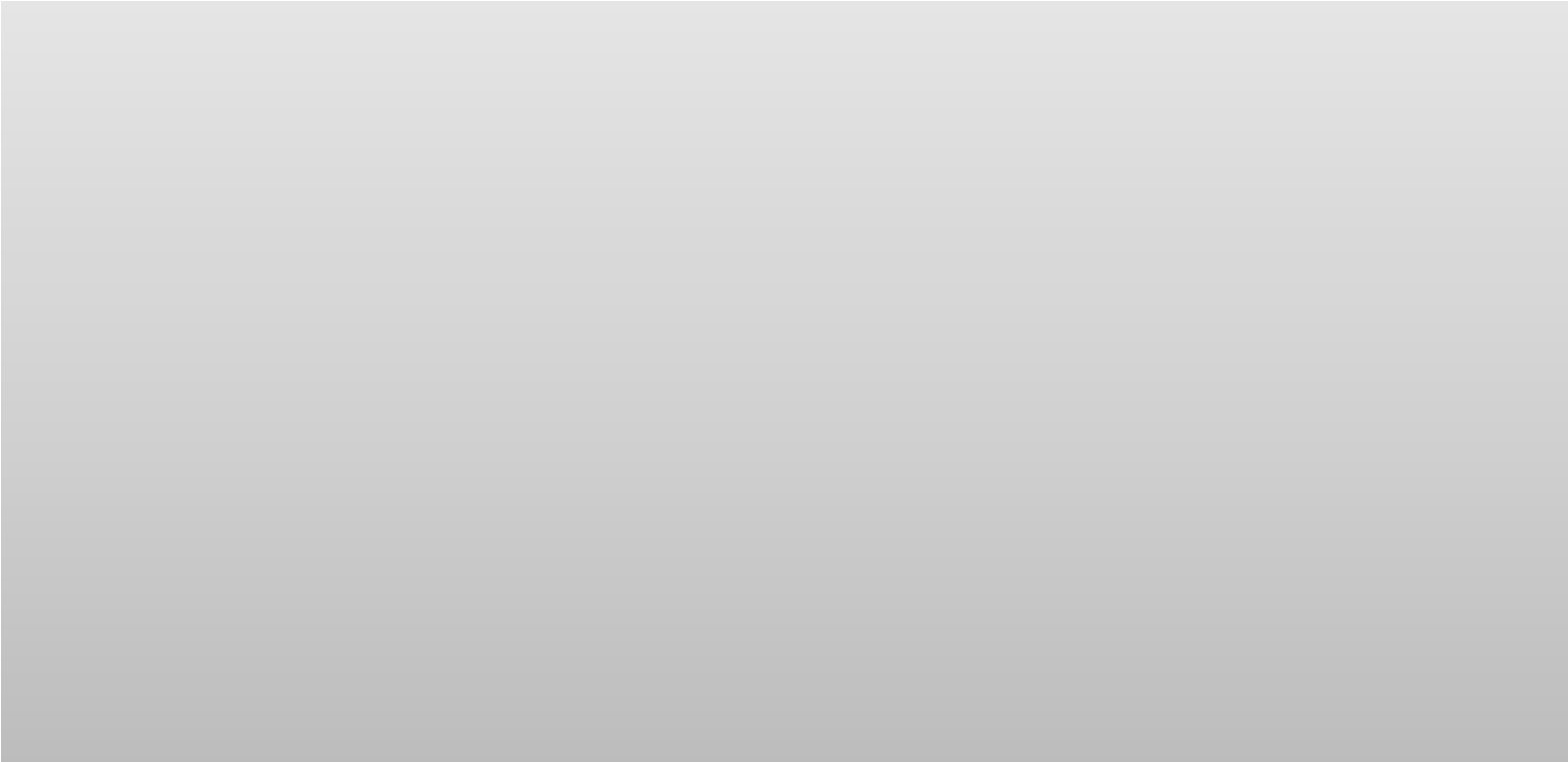


So, what should be fixed?



Enter the coordinates of A:

AX = 3



AY = 0

Enter the coordinates of B:

BX = 6

BY = 4

Enter the coordinates of C:

CX = -1

CY = 3

The straight lines have the dimensions:

3

0 6 4 -1

3

50

25

25

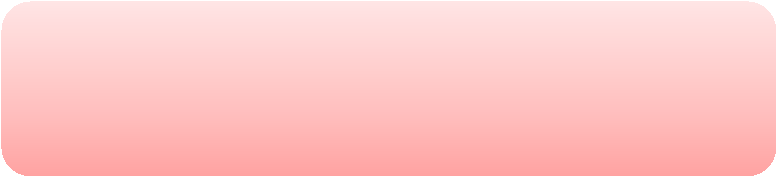
50

The points form an Isosceles triangle

The points form an Right Angled triangle

## A Practical Application

* Application: Game Design
* Create a ‘guess the number’ game using python



Design the game



Time: 20 mins+ 20 mins

## Solution: Pseudocode

Welcome the player to the game and explain it

Pick a random number between 1 and 100 Ask the player for a guess

Set the number of guesses to 1

While the player’s guess does not equal the number

If the guess is greater than the number Tell the player to guess lower

Otherwise

Tell the player to guess higher Get a new guess from the player

Increase the number of guesses by 1 Congratulate the player on guessing the number

Let the player know how many guesses it took

## Solution



# Guess My Number

import random

print("\tWelcometo 'Guess My Number'!")

print("\nI'mthinking of a number between 1 and 100.")

print("Try to guess it in as few attempts as possible.\n")

# set the initial values

the\_number= random.randint(1, 100)



guess =

int

(

input("Take a guess: "

))

tries = 1

# guessing loop

while guess != the\_number:

if guess > the\_number:

print("Lower...")

else:

print("Higher...")

guess = int(input("Take a guess: "))

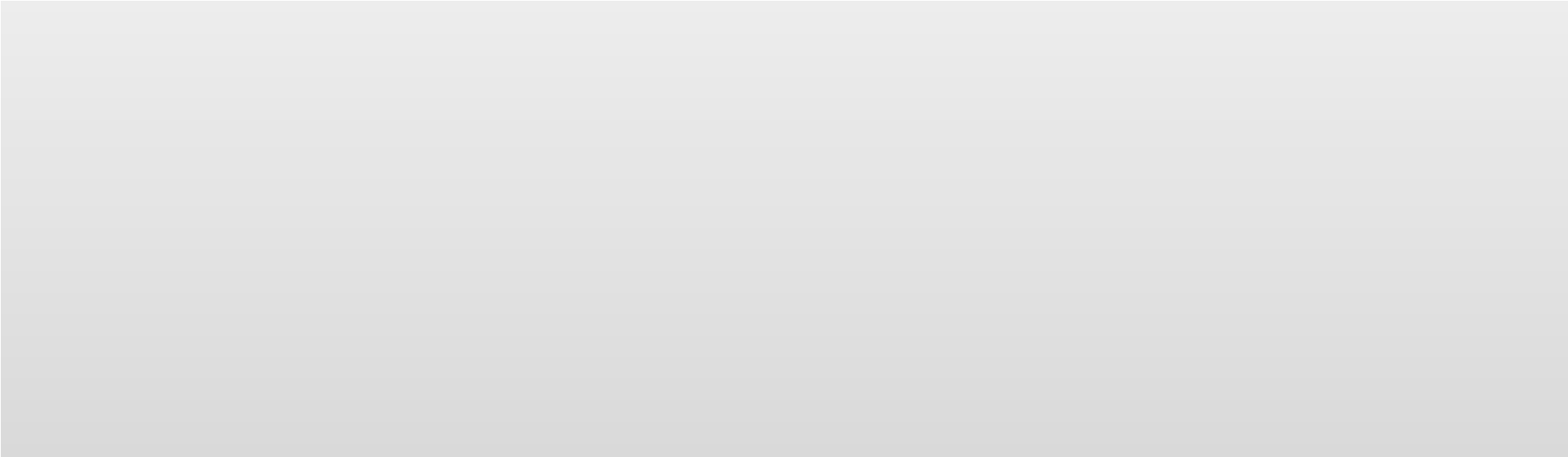
tries += 1

print("You guessed it! The number was", the\_number)

print("And it only took you", tries, "tries!\n")

input("\n\nPressthe enter key to exit.")

## Output



>>>

Welcome to 'Guess My Number'!

I'm thinking of a number between 1 and 100.

Try to guess it in as few attempts as possible.

Take a guess: 7

Higher...

Take a guess: 90



Lower...

Take a guess: 67

Lower...

Take a guess: 56

Lower...

Take a guess: 45

You guessed it! The number was 45

And it only took you 5 tries!

Press the enter key to exit.

## Challenge Problem #1

• Improvise the ‘guess my number’ game with different difficulty levels

## Challenge Problem #2

* Design a word jumble game
* The computer picks a random word from a group and then creates a jumbled version of it, where the letters are in random order. The player has to guess the original word to win the game.

## Challenge #3

• Write a python script to print the Floyd triangle for a given height in number of lines

## Challenge #4

• Write a python script to generate a pyramid of characters for a given height in number of lines

– Both character and height should be user inputs

## Challenge #5

• Given a string and a sub-string from the user, write a python script to find out how many occurrences of the given substring are present in the string

## Challenge #6

• Write a python script to display a calendar of a given month