ICT 133
Structured Programming

Seminar 2

Topics

- String data type and string formatting
- Decision/selection control structure

str Data Type

- Elements are characters enclosed within single ('), double (") or triple ("") quotes e.g.,

```
'Ann'
"Ann"
'''Ann'''
```

str Data Type

- Individual elements in a str is accessed through *indexing,*
  - numbered from the left, starting with 0
  - numbered from the right, starting with -1

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

Accessing individual elements

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

```
greet = "Hello Bob"
greet[0]                                    'H'
print(greet[0], greet[2], greet[4])         H l o
x = 8
print(greet[x - 2])                         B
print(greet[-3])                            B
```

Accessing Contiguous Elements

- Slicing: `[<start>:<end>:<increment>]`
- The slice contains the elements beginning at position `start` up to but NOT including the element at position `end`.
- If `<start>` (or `<end>`) is missing, then the start (or the end) of the sequence is used.
- If `<increment>` is missing, then 1 is used.

Accessing Contiguous Elements

H	e	l	l	o		B	o	b
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

```
greet[0:3]        'Hel'
greet[3:0:-1]     'lle'
greet[:5]         'Hello'
greet[5:]         ' Bob'
greet[:]          'Hello Bob'
greet[::-1]       'boB olleH'
```

Combining Elements

- *Concatenation* (+) "glues" two str together

  `"Hello" + "Bob"` evaluates to `'HelloBob'`
  `greet[0] + greet[-1]` evaluates to `'Hb'`

- *Repetition* (*) does a multiple concatenations of the str

  `greet[0:2]*3` evaluates to `'HeHeHe'`

function len

- Return the length or number of elements

```
len("spam")          4
len(greet)           ?
```

Summary

Operator	Meaning
+	Concatenation
*	Repetition
aStr[index]	Indexing
aStr[start:end:increment]	Slicing
len(<sequence>)	Length

Useful String Functions

- split(separator)

```
firstName, lastName = input("Enter first name
followed by last name:").split()

Enter first name followed by last name: John Tan
```

```
coords = input("Enter the point coordinates
(x,y):").split(",")
x,y = float(coords[0]), float(coords[1])

Enter the point coordinates  (x,y): 3.1, 5.3
```

More String Methods

`s.capitalize()`	Copy of s with only the first character capitalized
`s.lower()`	Copy of s with all characters in lowercase
`s.upper()`	Copy of s with all characters in uppercase
`s.title()`	Copy of s; first character of each word capitalized
`s.count(substr)`	Count the number of occurrences of substr in s

More String Methods

`s.center(width)`	Center s in a field of given width
`s.rjust(width)`	Like center, but s is right-justified
`s.ljust(width)`	Like center, but s is left-justified
`s.join(list)`	Concatenate list of strings into one large string using s as separator.
`s.lstrip()`	Copy of s with leading whitespace removed
`s.rstrip()`	Copy of s with trailing whitespace removed
`s.strip()`	Copy of s with leading and trailing whitespace removed

More String Methods

`s.count(substr)`	Count the number of occurrences of substr in s
`s.find(sub)`	Find the first position where sub occurs in s
`s.rfind(sub`	Like find, but returns the right-most position
`s.replace(oldsub, newsub)`	Replace occurrences of oldsub in s with newsub
`str(expr)`	Convert expr to string

String Formatting

- `<template-string>.format(<values>)`

```
"Total = ${0:0.2f}". format(total)
```

  - **{ } :** "slot" for value in format
  - format specifier

```
{0:0.2f}
<index>:<format-specifier>
```

<min width> <precision> <type>

String Formatting

0 1 2

```
"{1} {0} won ${2}" .format("Smith", "Mr.", 100)
'Mr. Smith won $100'
```

```
"Number {:5d}, min width 5 digits".format(7)
'Number     7, min width 5 digits'
```

```
"Number, {0:8.3f}, 3 dec places".format(3.1416)
'Number,    3.142, 3 dec places'
```

String Formatting

```
"left justification, min 5 characters:
{0:<5s}".format("Hi!")
'left justification, min 5 characters: Hi!   '

"right justification: {0:>5}.format("Hi!")
'right justification:   Hi!'

"centered: {0:^5}".format("Hi!")
'centered:  Hi! '
```

Control Structures

- Sequence (seminar 1)
  Each statement executes once, from top

- **Decision – branching, selection (seminar 2)**
  Each statement executes 0 time or once

- Iteration – loop, iteration, repetition (seminar 3)

Python selection statements

- `if` statement (one-way selection)
- `if else` statement (two-way selection)
- `if elif … else` statement (n-way selection)
- nested selection statements
- conditional expression

Introduction to Selection

- Problem
  - Roots of quadratic equation
  - If discriminator is less than zero
    - Output no real root
  - If discriminator is more than or equals to zero
    - Compute and print the roots

Relational operators

- >, <, >=, <=, ==, !=

- Result of comparison:
  - True, False (not yes, no)
  - bool data type

5 > 0	True	5 >= 0	?
-1 > 0	False	-1 <= 0	?

Relational operators

- greater than >
  greater than or equal >=

- less than <
  less than or equal <=

- equal ==
  not equal !=

One-way Selection

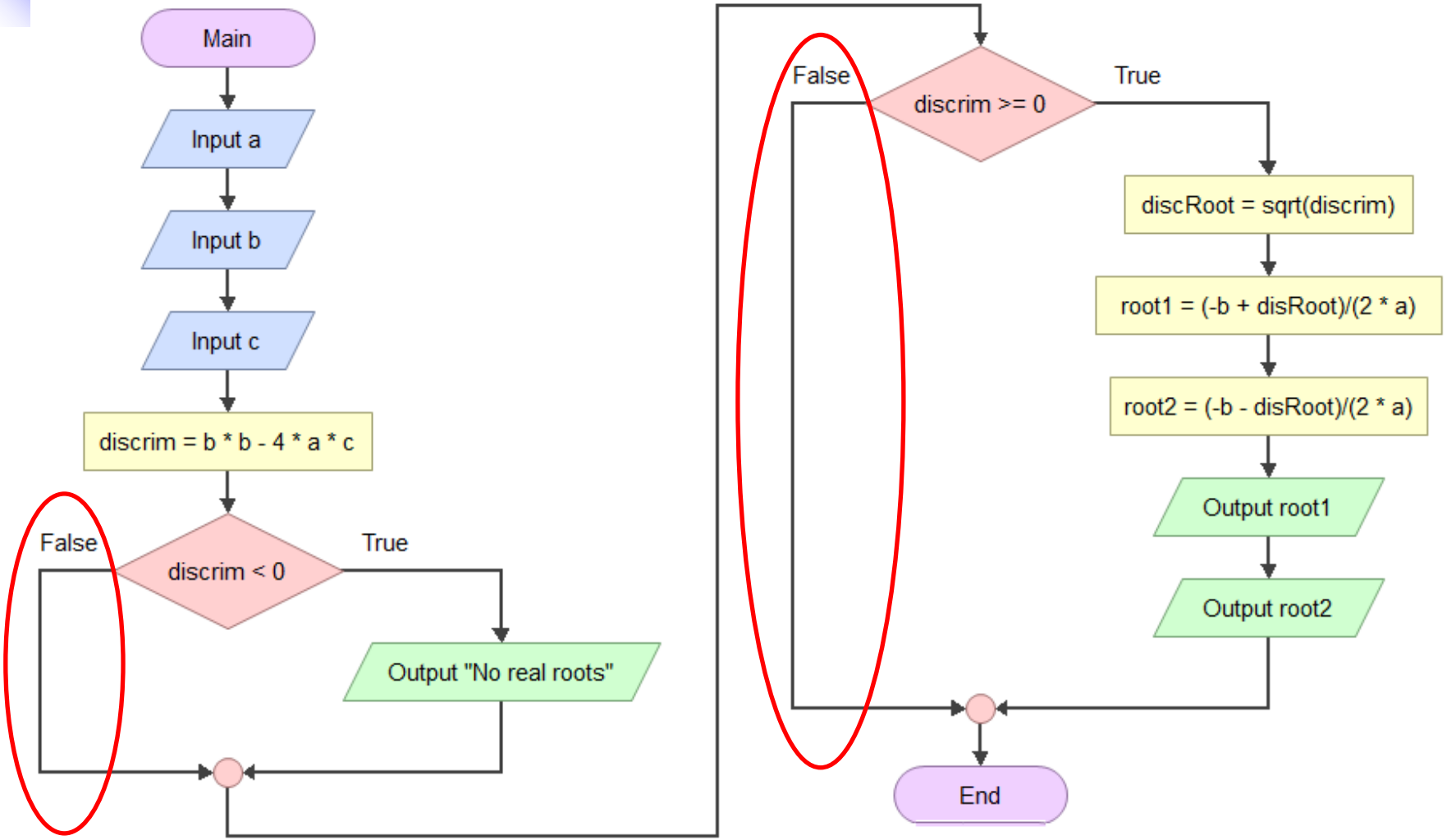
```
if <boolean expression>:
    <body>
```

- Boolean expression evaluates to either true / false
  - If result is true, statements in the body are executed.
- Control passes to the next statement after the `if`.

One-way Selection Example

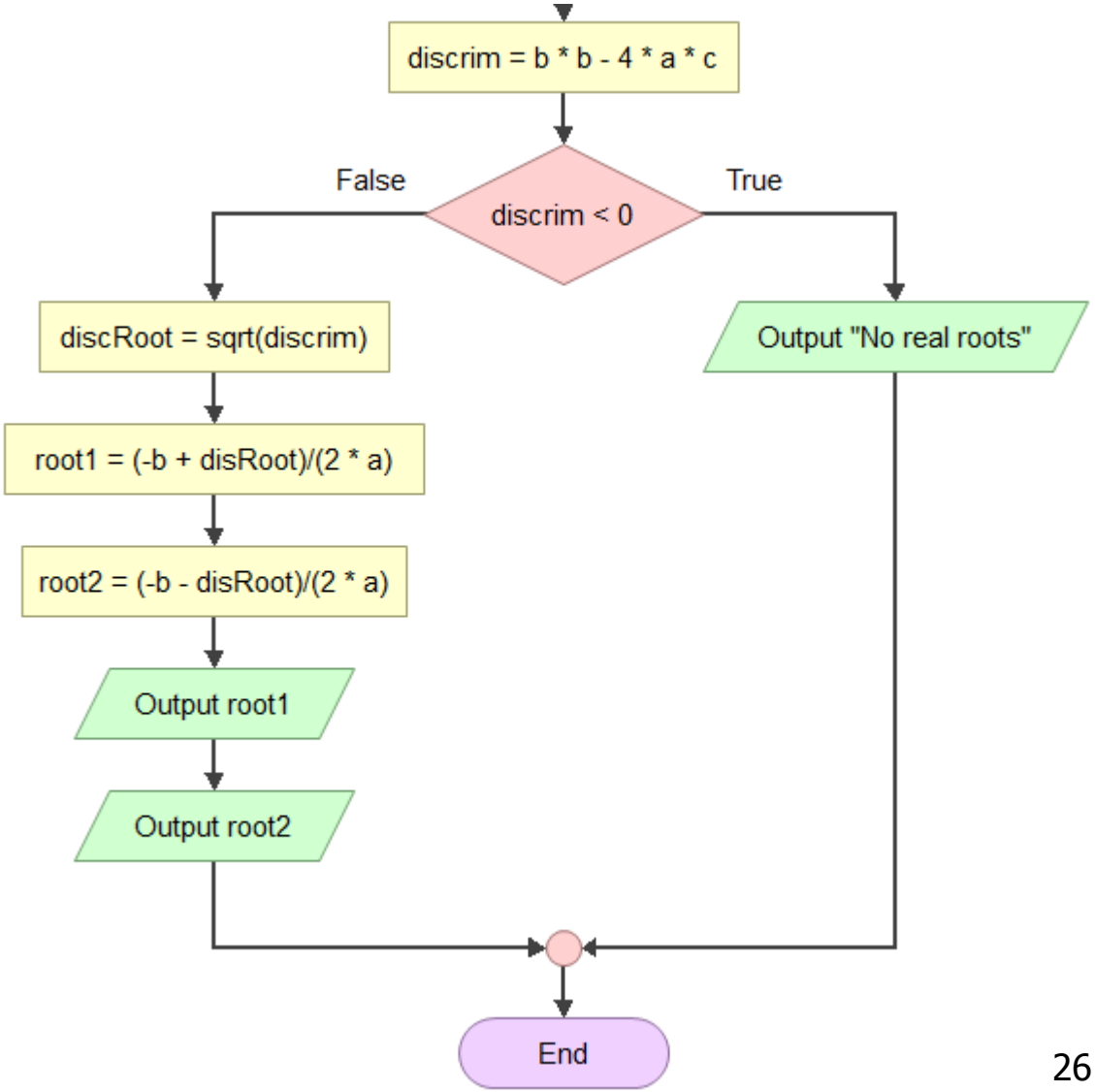
```
import math
def main():
    a = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient c: "))
    discrim = b * b - 4 * a * c
    if discrim < 0:
        print("No real roots")
    if discrim >= 0:
        discRoot = sqrt(discrim)
        root1 = (-b + disRoot) / (2 * a)
        root2 = (-b - disRoot) / (2 * a)
        print(root1, root2)
```

One-way Selection Example



Two-Way Selection

```
if <condition>:
    <statements>
else:
    <statements>
```

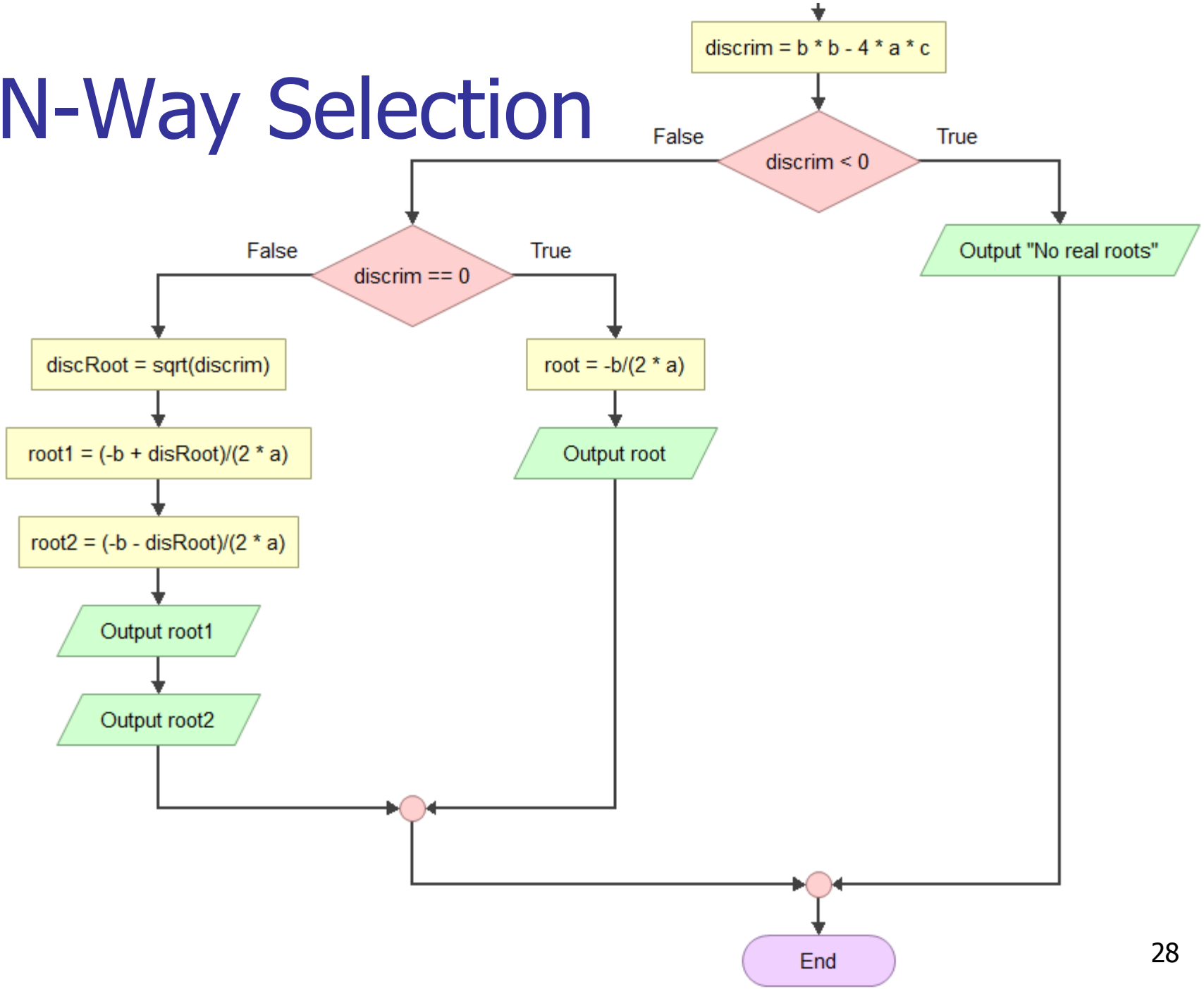


Two-Way Selection

```
import math

def main():
    a = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient c: "))
    discrim = b*b-4*a*c
    if discrim < 0:
        print("\nThe equation has no real roots!")
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print ("\nThe solutions are:", root1, root2 )
```

N-Way Selection



N-Way Selection

```
import math

def main():
    a = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient c: "))
    discrim = b*b-4*a*c
    if discrim < 0:
        print("\nThe equation has no real roots!")
    elif discrim == 0:
        root = -b / (2 * a)
        print("\nThere is a double root at", root)
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print ("\nThe solutions are:", root1, root2 )
```

String comparison

- Follow the Ascii table
  - Ascii code for ‘A’ is 65, ‘a’ is 97
  - Therefore,

    if ‘A’ != ‘a’:

    evaluates to True

- Based on length
  - Therefore,

    if ‘A’ >= ‘A1’:

    evaluates to False

Nested Selection

- Nested selection statements are selection statement within selection statement

- You can nest as many if…else statements as you want

```
if <cond1>:
    if <cond2>:
        s1
    elif <cond3>:
        s2
    else:
        s3
else:
   s4
```

Example of Nested Selection

```
if gender == ‘m’:
    if age < 18:
        print(‘Not enlisted yet’)
    elif age == 18:
        print(‘To be enlisted’)
    else:
        print(‘Already enlisted’)
else:
   print(‘No need to be enlisted’)
```

Nested within if

Logical Operator – and

P	Q	P **and** Q
Price is low	*Book is interesting*	*Then Buy book*
True	True	True
True	False	False
False	True	False
False	False	False

if price is low and the book is interesting

then

buy book

Logical Operator - or

P	Q	P or Q
Very sunny	*Raining*	*Then Bring umbrella*
True	True	True
True	False	True
False	True	True
False	False	False

if it is very sunny or it is raining

then

bring umbrella

Logical Operator - not

P	not P
Raining	*Not raining* *Then Outdoor activity*
T	F
F	T

if not raining

then

Outdoor activity

Precedence of Logical Operators

Operator	Description	Precedence
or	Boolean OR	lowest
and	Boolean AND	Next highest
not x	Boolean NOT	highest

Consider

```
a or not b and c
```

This statement is equivalent to

```
(a or ((not b) and c))
```

Example: Using logical operators

- Display whether a given integer is
  - positive even
  - Negative even
  - Positive odd
  - Negative odd

Solution 1: Nested and without logical operator

```
num = ?
if num%2==0:
    if num >= 0:
        print("+ve even")
    else:
        print("-ve even")
else:
    if num >=0:
        print("+ve odd")
    else:
        print("-ve odd")
```

Solution 2: Not nested and with logical operator

```
num = ?
if num%2==0 and num >= 0:
    print("+ve even")
elif num%2==0 and num < 0:
    print("-ve even")
elif num%2 != 0 and num >=0:
    print("+ve odd")
elif num%2 != 0 and num < 0:
    print("-ve odd")
```

Solution 3: Not nested and with logical operator

```
num = ?
if num%2==0 and num >= 0:
    print("+ve even")
elif num%2==0:
    print("-ve even")
elif num >=0:
    print("+ve odd")
else:
    print("-ve odd")
```

Solution 4: Not nested and without logical operator

```
num = ?
if num >= 0:
    print("+ve", end = " ")
else:
    print("-ve", end = " ")

if num%2==0:
    print("even")
else:
    print("odd")
```

Solution 5: Conditional expressions

- Syntax:

```
value1 if condition else value2
```

num = ?
sign = *"+ve" if num >= 0 else "-ve"*
evenOrOdd = *"even" if num%2==0 else "odd"*
print(sign, evenOrOdd)

Precedence of operators

Operator		Description
if – else		Conditional expression
or		Boolean OR
and		Boolean AND
not x		Boolean NOT
<, <=, >, >=, !=, ==	Increasing precedence	Comparisons, relational operators
+, -		Addition and subtraction
*, /, //, %		Multiplication, division, floor division, remainder
+x, -x		Positive, negative
**		Exponentiation
x[index], x[index:index], x(arguments...)		Subscription, slicing, function call

Shortcut Boolean expression

- `0 <= mark < 50` is the same as
  `0 <= mark and mark < 50`

- This shortcut is incorrect
  `if gender == "m" or "M":`

- `if gender in "mM":` is the same as
  `if gender == "m" or gender == "M":`

Redundant comparisons

- Comparing bool variables

```
raining = true
if  raining == True:  # == True is redundant
    print("Bring umbrella!")
```

- bool variables already evaluate to True/False. Use this:

```
if raining:
   print("Bring umbrella!")
```

Redundant comparisons

```
if  x > 0:
   print("x is positive")
elif x <= 0: # redundant
    print("x is not positive")
```

Do this instead:

```
if  x > 0:
   print("x is positive")
else:
    print("x is not positive")
```

Redundant comparisons

```
if  mark >= 0 and mark <50:
    print("fail")
elif mark >=50 and mark <=70:
   print("Credit")
elif mark > 70 and mark <= 100:
   print("Distinction")
```

Where?