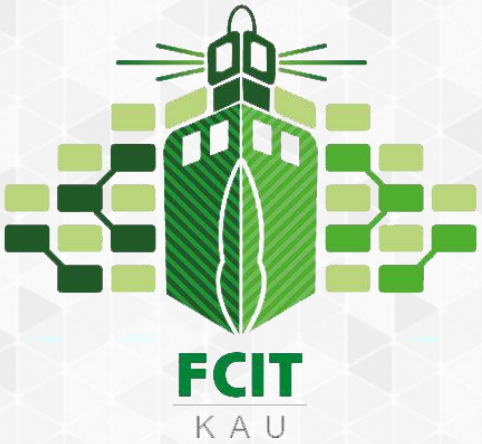
FACULTY OF COMPUTING
& INFORMATION TECHNOLOGY
KING ABDULAZIZ UNIVERSITY



كلية الحاسبات
وتقنية المعلومات
جامعة الملك عبدالعزيز

Chapter 4
Selections

CPIT 110 (Problem-Solving and Programming)

Introduction to Programming Using Python, By: Y. Daniel Liang

Sections

Python Online IDE

Programs

- Program 1: Math Learning Tool
- Program 2: Simple if Demo
- Program 3: Improved Math Learning Tool
- Program 4: Chinese Zodiac
- Program 5: Computing BMI
- Program 6: Test Boolean Operators
- Program 7: Leap Year
- Program 8: Lottery

Check Points

- Section 4.3
  - #1
  - #2
  - #3
  - #4
- Section 4.4
  - #5
- Section 4.6
  - #6
  - #7
- Section 4.7
  - #8
  - #9
  - #10
- Section 4.8
  - #11
  - #12
  - #13
- Section 4.11
  - #14
  - #15
  - #16
  - #17
  - #18
  - #19
  - #20
  - #21
- Section 4.14
  - #22
  - #23
  - #24
  - #25
  - #26
  - #27
- Section 4.15
  - #28
  - #29

Objectives

- To write Boolean expressions by using comparison operators (4.2).
- To generate random numbers by using the **random.randint(a, b) or random.random()** functions (4.3).
- To program with Boolean expressions (**AdditionQuiz**) (4.3).
- To implement selection control by using one-way **if** statements (4.4)
- To implement selection control by using two-way **if .. else** statements (4.6).
- To implement selection control with nested **if ... elif ... else** statements (4.7).
- To avoid common errors in **if** statements (4.8).
- To program with selection statements (4.9).
- To combine conditions by using logical operators (**and**, **or**, and **not**) (4.11).
- To use selection statements with combined conditions (**LeapYear**, **Lottery**) (4.12 – 4.13).
- To write expressions that use the conditional expressions (4.14).
- To understand the rules governing operator precedence and associativity (4.15).

4.1. Motivations

Motivations

- Consider this problem from Chapter 2:

LISTING 2.2 ComputeAreaWithConsoleInput.py

```
1  # Prompt the user to enter a radius
2  radius = eval(input("Enter a value for radius: "))
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius " , radius , " is " , area)
```

Motivations

- What happens if the user inputs a negative value for radius?
  - The program would print an invalid result.
- If the radius is negative, you don't want the program to compute the area.
- **How can you deal with this situation?**

Motivations

- Python provides selection statements, which allow you to choose actions based on certain conditions.
- For example, the following selection statement could be used in the previous program:

ModifiedComputeAreaWithConsoleInput.py

```
1  # Prompt the user to enter a radius
2  radius = eval(input("Enter a value for radius: "))
3
4  if radius < 0:
5      # radius is negative, so print an error message
6      print("Incorrect input")
7  else:
8      # Compute area
9      area = radius * radius * 3.14159
10     # Display results
11     print("The area for the circle of radius " , radius , " is " , area)
```

Run

Motivations

ModifiedComputeAreaWithConsoleInput.py

```
1  # Prompt the user to enter a radius
2  radius = eval(input("Enter a value for radius: "))
3
4  if radius < 0:
5      # radius is negative, so print an error message
6      print("Incorrect input")
7  else:
8      # Compute area
9      area = radius * radius * 3.14159
10     # Display results
11     print("The area for the circle of radius " , radius , " is " , area)
```

Run

```
Enter a value for radius: -5  <Enter>
Incorrect input
```

```
Enter a value for radius: 0  <Enter>
The area for the circle of radius  0  is  0.0
```

```
Enter a value for radius: 5<Enter>
The area for the circle of radius  5  is  78.53975
```

Motivations

- Selection statements use conditions to test if something is true or false.
- These conditions are known as Boolean expressions.
- A Boolean expression is an expression that evaluates to a Boolean value (True or False).
- This chapter introduces Boolean types, values, comparison operators, and expressions.

4.2. Boolean Types, Values, and Expressions

- Boolean Data Types
- Relational Operators
- Boolean Variables
- Convert Boolean Value to Integer
- Convert Numeric Value to Boolean
- Puzzle

Boolean Data Types

- Often in a program you need to compare two values, such as whether x is greater than y.
  - Or if an inputted value, such as radius, is less than zero.
- There are six comparison operators (also known as relational operators) that can be used to compare two values. The result of the comparison is a Boolean value: True or False.
- For example:

```
1  x = 10 > 20
2  y = 10 < 20
3  print ("x =", x) # output: False
4  print ("y =", y) # output: True
```

```
x = False
y = True
```

Relational Operators

TABLE 4.1 Comparison Operators

Python Operator	*Mathematics Symbol*	*Name*	*Example* (radius is 5)	*Result*
`<`	<	less than	`radius < 0`	`False`
`<=`	≤	less than or equal to	`radius <= 0`	`False`
`>`	>	greater than	`radius > 0`	`True`
`>=`	≥	greater than or equal to	`radius >= 0`	`True`
`==`	=	equal to	`radius == 0`	`False`
`!=`	≠	not equal to	`radius != 0`	`True`

Caution

- The equal to comparison operator is two equal signs (==), not a single equal sign (=). The latter symbol is for assignment.


```
>>> 20 == 30
False
>>> 50 == 50
True
>>> 50.0 == 50
True
>>> 50 = 30
SyntaxError: can't assign to literal
>>> 60.0001 == 60
False
```

Boolean Variables

- A variable that holds a Boolean value is known as a Boolean variable.
- The Boolean data type is used to represent Boolean values (True or False).
- A Boolean variable can hold one of the two values: True or False.
- For example, the following statement assigns the value True to the variable lightsOn:

```
lightsOn = True
```

- True and False are literals, just like a number such as 10. They are reserved words and cannot be used as identifiers in a program.

Convert Boolean Value to Integer

- Internally, Python uses 1 to represent True and 0 for False.
- You can use the int function to convert a Boolean value to an integer.


```
>>> int(True)
1
>>> int(False)
0
>>> print(int(True))
1
>>> print(int(20 > 50))
0
>>> print(int(True == False))
0
>>> print(int(20 * 2 > 30))
1
```

Convert Numeric Value to Boolean

- You can also use the bool function to convert a numeric value to a Boolean value.
- The function returns False if the value is 0; otherwise, it always returns True.


```
>>> bool(1)
True
>>> bool(0)
False
>>> bool(15)
True
>>> bool(-20)
True
>>> bool(10 + 10 - 20)
False
```

Puzzle

- What is the output of the following script?

```
1  x = int(True) + int(True) + int(int(True) - int(True))
2  y = int(True) * int(bool(50) * 3)
3  r = x + y
4  print("x = ", x)
5  print("y = ", y)
6  print("r = ", r)
7  print("bool(r) = ", bool(r))
8  print("bool(r - r) = ", bool(r - r))
```

```
x =  2
y =  3
r =  5
bool(r) =  True
bool(r - r) =  False
```

4.3. Generating Random Numbers

- Generating Random Integer Numbers
- Program 1: Math Learning Tool
- randrange function
- Generating Random Float Numbers
- Check Point #1 - #4

Generating Random Integer Numbers
randint function

- To generate a random number, you can use the randint(a, b) function in the random module.
- This function returns a random integer between a and b, inclusively.
- To obtain a random integer between 0 and 9, use randint(0, 9):

```
1  import random
2  x = random.randint(0, 9)
3  # x could be 0,1,2,3,4,5,6,7,8, or 9
4  print(x)
```

```
x = 2
```

```
x = 9
```

```
x = 7
```

```
x = 3
```

Math Learning Tool
Program 1

Write a program that helps a first-grader practice addition. The program should randomly generate two single-digit integers and should then ask the user for the answer. The program will then display a message stating if the answer is true or false.

```
What is 9 + 4? 12  <Enter>
9 + 4 = 12 is False
```

```
What is 3 + 1? 4  <Enter>
3 + 1 = 4 is True
```

Math Learning Tool
Phase 1: Problem-solving

- What does “single-digit integers” mean?

0 - 9

1-digit (single) integer → [0 - 9]

```
1 import random
2 number = random.randint(0, 9)
```

0 – 9	0 – 9

2-digit integer → [10 - 99]

```
1 import random
2 number = random.randint(10, 99)
```

0 – 9	0 – 9	0 – 9

3-digit integer → [100 - 999]

```
1 import random
2 number = random.randint(100, 999)
```

Math Learning Tool
Phase 1: Problem-solving

Design your algorithm:

1. Generate two single-digit integers for number1 and number2.
   - Use randint(0, 9)
   - Example: number1 = 2 and number2 = 6
2. Ask the user to answer a question
   - Example: “What is 2 + 6 ?”
3. Print whether the answer is true or false

Math Learning Tool
Phase 2: Implementation

LISTING 4.1 AdditionQuiz.py

```
1  import random
2
3  # Generate random numbers
4  number1 = random.randint(0, 9)
5  number2 = random.randint(0, 9)
6
7  # Prompt the user to enter an answer
8  answer = eval(input("What is " + str(number1) + " + "
9                      + str(number2) + "? "))
10
11 # Display result
12 print(number1, "+", number2, "=", answer,
13       "is", number1 + number2 == answer)
```

Run

```
What is 1 + 7? 8   <Enter>
1 + 7 = 8 is True
```

```
What is 4 + 8? 9   <Enter>
4 + 8 = 9 is False
```

Math Learning Tool
Trace The Program Execution

```
What is 4 + 8? 9   <Enter>
4 + 8 = 9 is False
```

line#	number1	number2	answer	output
4	4			
5		8		
8			9	
12				4 + 8 = 9 is False

Math Learning Tool
Discussion

- The program uses the randint function defined in the random module.
- The import statement imports the module (line 1).
- Lines 4-5 generate two numbers, number1 and number2.
- Line 8 obtains an answer from the user.
- The answer is graded in line 12 using a Boolean expression number1 + number2 == answer.

randrange function

- Python also provides another function, randrange(a, b), for generating a random integer between a and b – 1, which is equivalent to randint(a, b – 1).
- For example, randrange(0, 10) and randint(0, 9) are the same.
- Since randint is more intuitive, the book generally uses randint in the examples.


```
>>> import random
>>> random.randrange(1, 3) # the value could be: 1 or 2
1
>>> random.randint(1, 3) # the value could be: 1, 2 or 3
2
>>> random.randint(0, 1) # the value could be: 0 or 1
1
>>> random.randrange(0, 1) # This will always be 0
0
```

Generating Random Float Numbers
random function

- You can also use the random() function to generate a random float r such that $\mathbf{0} \leq \boldsymbol{r} < \mathbf{1}$
- For example:


```
>>> import random
>>> random.random()
0.3362800598715141
>>> random.random()
0.886713208073315
>>> random.random()
0.9735731618649538
```

- Note: the random() function returns a random float number between **0.0** and **1.0** (excluding 1.0).

Check Point #1

How you can generate a float number with n-digit before the decimal point?

- We can do that by multiplying the generated number with 10^n. For example:

Python

```
>>> import random
>>> random.random() * 10 ** 1 # 1-digit float number
8.573088600232266
>>> random.random() * 10 ** 2 # 2-digits float number
99.56489589285628
>>> random.random() * 10 ** 3 # 3-digit float number
428.6688384440885
```

- Formula:

```
number = random.random() * 10 ** n
```

n for the number of the digits before the decimal point

Check Point #2

How you can generate a float number with n-digit before the decimal point and d-digit after the decimal point?

- We can do that by using the round function as the following:

Python

```
>>> import random
>>> round(random.random() * 10 ** 1, 2)
7.66
>>> round(random.random() * 10 ** 2, 2)
79.95
>>> round(random.random() * 10 ** 4, 3)
2969.055
```

- Formula:

```
number = round(random.random() * 10 ** n, d)
```

n for the number of the digits before the decimal point
d for the number of the digits after the decimal point

Check Point #3

How you can generate a random float number that is equal or greater than a and less than b ($a \leq number < b$).

- We can do that as the following:

```
number = a + (random.random() * (b - a))
```

- Examples:

Python

```
>>> import random
>>> a, b = 1, 3
>>> a + (random.random() * (b - a)) # a = 1, b = 3
1.6393718672389215
>>> a, b = 10, 20
>>> a + (random.random() * (b - a)) # a = 10, b = 20
15.046056155663972
```

Check Point #4

How do you generate a random integer i such that $0 \leq i < 20$?

```
1  import random
2  i = random.randint(0, 19)
3  # Or ->
4  i = random.randrange(0, 20)
```

How do you generate a random integer i such that $10 \leq i \leq 50$?

```
1  import random
2  i = random.randint(10, 50)
3  # Or ->
4  i = random.randrange(10, 51)
```

4.4. if Statements

- Types of Selection Statements
- One-way if Statements
- if Block
- Program 2: Simple if Demo
- Check Point #5

Types of Selection Statements

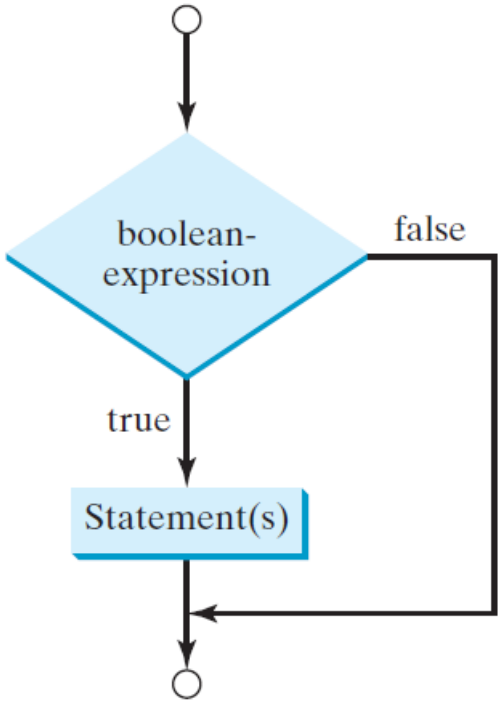
- The preceding program (Program 1) displays a message such as 6 + 2 = 7 is False. If you wish the message to be 6 + 2 = 7 is incorrect, you have to use a selection statement to make this minor change.
- Python has several types of selection statements:
  - one-way if statements
  - two-way if-else statements
  - nested if statements
  - multi-way if-elif-else statements
  - conditional expressions

One-way if Statements

- A one-way if statement executes an action if and only if the condition is true.
- The syntax for a one-way if statement is:

```
if boolean-expression:
    statement(s)
```

- The flowchart to the right demonstrates the syntax of an if statement.
- Example:




```
1 lightOn = True
2
3 if lightOn:
4     print("Light ON")
```

Note

- Also, the following syntax is valid for one-way if statement with a one statement on one line:

```
if boolean-expression: statement
```

- Example:

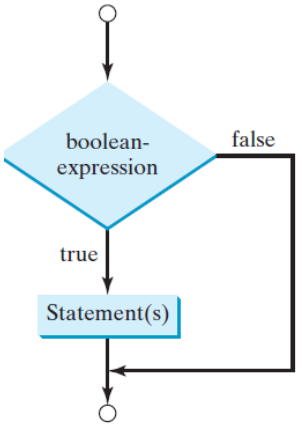
```
1  lightOn = True
2  if lightOn: print("Light ON")
3  if lightOn == False: print("Light OFF")
```

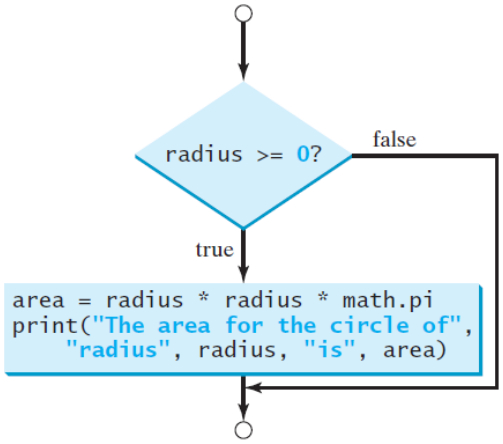
  - It is equivalent to:

```
1  lightOn = True
2
3  if lightOn:
4      print("Light ON")
5
6  if lightOn == False:
7      print("Light OFF")
```

Remember

- A flowchart is a diagram that describes an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting these with arrows.
- Process operations are represented in these boxes, and arrows connecting them show flow of control.
- A diamond box is used to denote a Boolean condition and a rectangle box is for representing statements.





if Block

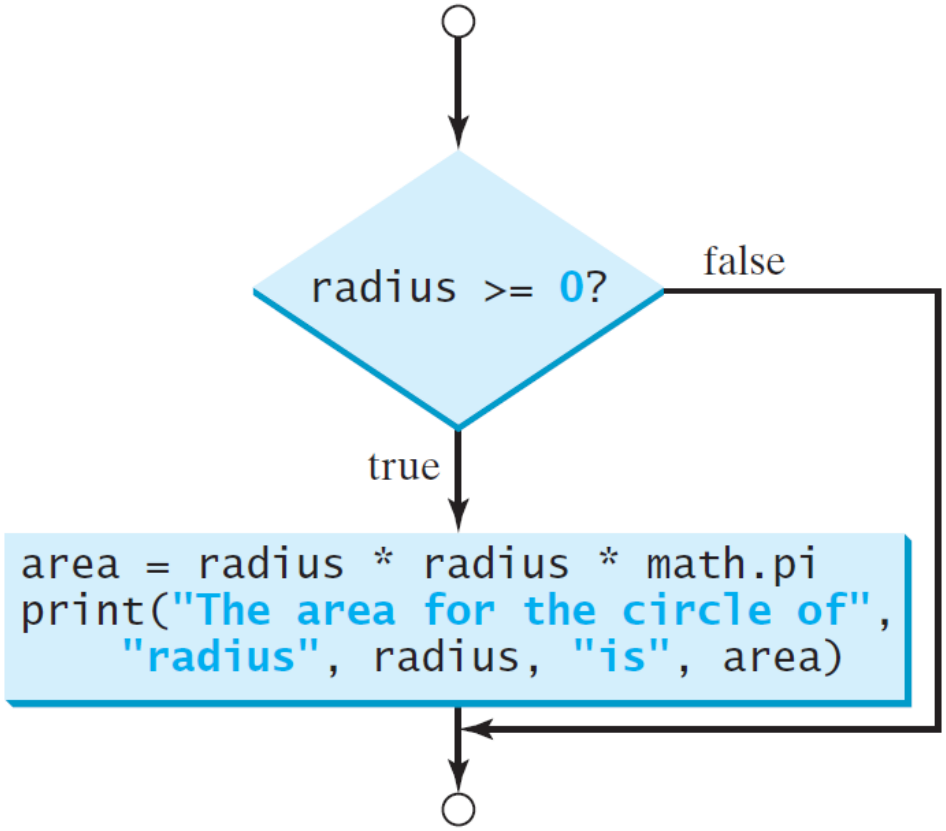
- If the boolean-expression evaluates to true, the statements in the if block are executed.
- The if block contains the statements indented after the if statement.
- For example:

```
if radius >= 0:
    area = radius * radius * math.pi
    print("The area for the circle of radius", radius, "is", area)
```

  - If the value of radius is greater than or equal to **0**, then the area is computed and the result is displayed; otherwise, these statements in the block are not executed.

if Block

```
if radius >= 0:
    area = radius * radius * math.pi
    print("The area for the circle of radius", radius, "is", area)
```



Note

- The statements in the if block must be indented in the lines after the if line and each statement must be indented using the same number of spaces.
- For example, the following code is wrong, because the print statement in line 3 is not indented using the same number of spaces as the statement for computing area in line 2.

```
1 if radius >= 0:
2     area = radius * radius * math.pi
3    print("The area for the circle of radius", radius, "is", area)
```

Note

```
1 if i > 0:
2 print("i is positive.")
```

(A) Wrong ✗

```
1 if i > 0:
2     print("i is positive.")
```

(B) Correct ✓

```
1 if i > 0: print("Positive")
2  print("Not in the if block")
```

(C) Wrong ✗

```
1 if i > 0: print("Positive")
2 print("Not in the if block")
```

(D) Correct ✓

```
1 if i > 0:
2     print("Positive")
3  print("Not in the if block")
```

(E) Wrong ✗

```
1 if i > 0:
2     print("Positive")
3 print("Not in the if block")
```

(F) Correct ✓

```
1 if i > 0      # missing :
2     print("i is positive.")
```

(G) Wrong ✗

```
1 if i > 0     :
2     print("i is positive.")
```

(H) Correct ✓

Simple if Demo
Program 2

Write a program that prompts the user to enter an integer. If the number is a multiple of 5, the program displays the result HiFive. If the number is divisible by 2, the program displays HiEven.

```
Enter an integer: 4   <Enter>
HiEven
```

```
Enter an integer: 15   <Enter>
HiFive
```

```
Enter an integer: 30   <Enter>
HiFive
HiEven
```

Simple if Demo
Phase 1: Problem-solving

Design your algorithm:

1. Prompt the user to enter an integer (number).
2. If number is a multiple of 5, print HiFive.
   - if (number % 5 == 0)
3. If number is divisible by 2, print HiEven.
   - if (number % 2 == 0)

Simple if Demo
Phase 2: Implementation

LISTING 4.1 SimpleIfDemo.py

```
1  number = eval(input("Enter an integer: "))
2
3  if number % 5 == 0:
4      print("HiFive")
5
6  if number % 2 == 0:
7      print("HiEven")
```

Run

```
Enter an integer: 4  <Enter>
HiEven
```

```
Enter an integer: 15  <Enter>
HiFive
```

```
Enter an integer: 30  <Enter>
HiFive
HiEven
```

Check Point #5

Write an if statement that assigns **1** to x if y is greater than **0**.

```
1  if y > 0:
2      x = 1
```

Write an if statement that increases pay by **3%** if score is greater than **90**.

```
1  if score > 90:
2      pay = pay + ( pay * (3 / 100) )
```

4.6. Two-Way if-else Statements

- Program 3: Improved Math Learning Tool
- Check Point #6 - #7

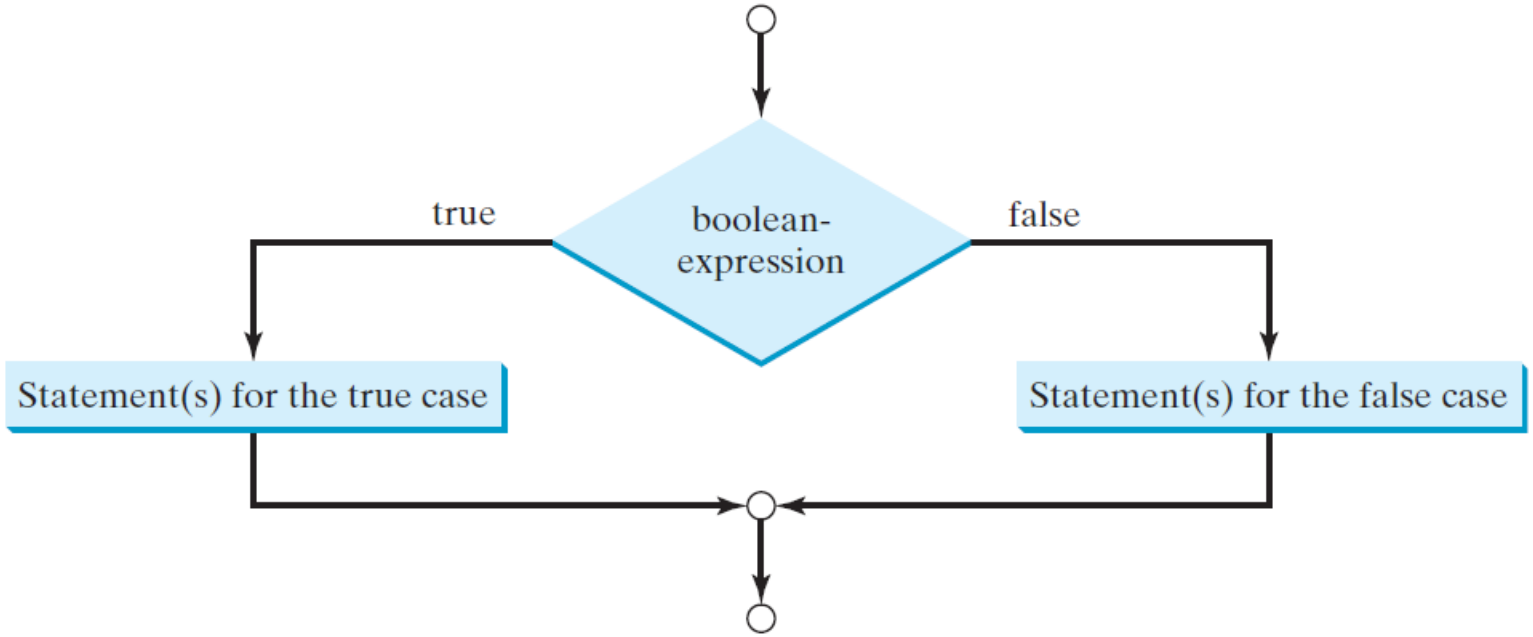
Two-way if-else Statement

- A one-way if statement takes an action if the specified condition is True.
  - If the condition is False, nothing is done.
- But what if you want to take one or more alternative actions when the condition is False?
- Answer: you can use a two-way if-else statement.
- The actions that a two-way if-else statement specifies differ based on whether the condition is True or False.

Two-way if-else Statement

- Here is the syntax for a two-way if-else statement:

```
if boolean-expression:
    statement(s)-for-the-true-case
else:
    statement(s)-for-the-false-case
```



Two-way if-else Statement

- If the boolean-expression evaluates to True, the statement(s) for the True case are executed.
- Else, the statement(s) for the False case are executed.
- For example, consider the following code:

```
1 if radius >= 0:
2     area = radius * radius * math.pi
3     print("The area for the circle of radius", radius, "is", area)
4 else:
5     print("Negative input")
```

  - If radius >= 0 is True, area is computed and displayed.
  - if it is False, the message Negative input is displayed.

Two-way if-else Statement

- Another example: this one determines whether a number is even or odd, as follows:

```
1 if number % 2 == 0:
2     print(number, "is even.")
3 else:
4     print(number, "is odd.")
```

Note

- The following syntax is valid for two way if-else statement with a one statement for the true case and a one statement for the false case.

```
if boolean-expression: statement-for-the-true-case
else: statement-for-the-false-case
```

- Example:

```
1 number = eval(input("Enter a number: "))
2 if number % 2 == 0: print(number, "is even.")
3 else: print(number, "is odd.")
```

Improved Math Learning Tool
Program 3

Write a program that helps a first-grader practice subtraction. The program should randomly generate two single-digit integers, number1 and number2, with number1 >= number2 and should then ask the user for the answer. The program will then display a message stating if the answer is correct. If wrong, the program should display the correct answer.

```
What is 6 - 6? 0   <Enter>
You are correct!
```

```
What is 9 - 2? 5   <Enter>
Your answer is wrong.
9 - 2 is 7
```

Improved Math Learning Tool
Phase 1: Problem-solving

Design your algorithm:

1. Generate two single-digit integers for number1 and number2.
   - Example: number1 = 6 and number2 = 2
2. If number1 < number2, swap number1 with number2.
   - Example: make number1 = 2 and number2 = 6
3. Ask the user to answer a question
   - Example: "What is 6 - 2 ?"
4. Print whether the answer is true or false
   - If the answer is false, print the correct answer

Improved Math Learning Tool
Phase 2: Implementation

LISTING 4.4 SubtractionQuiz.py

Run

```
1  import random
2
3  # 1. Generate two random single-digit integers
4  number1 = random.randint(0, 9)
5  number2 = random.randint(0, 9)
6
7  # 2. If number1 < number2, swap number1 with number2
8  if number1 < number2:
9      number1, number2 = number2, number1 # Simultaneous assignment
10
11 # 4. Prompt the student to answer "what is number1 - number2?"
12 answer = eval(input("What is " + str(number1) + " - " +
13     str(number2) + "? "))
14
15 # 4. Grade the answer and display the result
16 if number1 - number2 == answer:
17     print("You are correct!")
18 else:
19     print("Your answer is wrong.\n", number1, "-",
20         number2, "is", number1 - number2)
```

Improved Math Learning Tool
Trace The Program Execution

```
What is 9 - 2? 5   <Enter>
Your answer is wrong.
9 - 2 is 7
```

line#	number1	number2	answer	output
4	2			
5		9		
9	9	2		
12			5	
19				Your answer is wrong. 9 - 2 is 7

Check Point #6

Write an if statement that increases pay by **3%** if score is greater than **90**, otherwise it increases pay by **1%**.

```
1  if score > 90:
2      pay = pay + ( pay * (3 / 100) )
3  else:
4      pay = pay + ( pay * (1 / 100) )
```

Check Point #7

What is the printout of the code in (a) and (b) if number is 30 and 35, respectively?

```
1  if number % 2 == 0:
2      print(number, "is even.")
3
4  print(number, "is odd.")
```

(a)

```
1  if number % 2 == 0:
2      print(number, "is even.")
3  else:
4      print(number, "is odd.")
```

(b)

```
30 is even.
30 is odd.
```

```
35 is odd.
```

```
30 is even.
```

```
35 is odd.
```

4.7. Nested if and Multi-Way if-elif-else Statements

- Nested if
- Nested if and Multi-Way if-elif-else Statements
- Trace if-elif-else Statement
- Program 4: Chinese Zodiac
- Check Point #8 - #10

Nested if

- The statement in an if or if-else statement can be any legal Python statement.
  - Including another if or if-else statement.
- The inner if statement is said to be nested inside the outer if statement.
- Example:

The **if j > k statement** is nested inside the **if i > k statement**

```
1  if i > k:
2      if j > k:
3          print("i and j are greater than k")
4  else:
5      print("i is less than or equal to k")
```

Nested if

- More details:
  - The inner if statement can contain another if statement.
  - In fact, there is no limit to the depth of the nesting.
- So what is the purpose?
  - The nested if statement can be used to implement multiple alternatives.
- Consider the following example in the next slide, which prints a letter grade according to the final number grade.

Nested if and Multi-Way if-elif-else Statements

```
1  if score >= 90.0:
2      grade = 'A'
3  else:
4      if score >= 80.0:
5          grade = 'B'
6      else:
7          if score >= 70.0:
8              grade = 'C'
9          else:
10             if score >= 60.0:
11                 grade = 'D'
12             else:
13                 grade = 'F'
```

(a)

equivalent

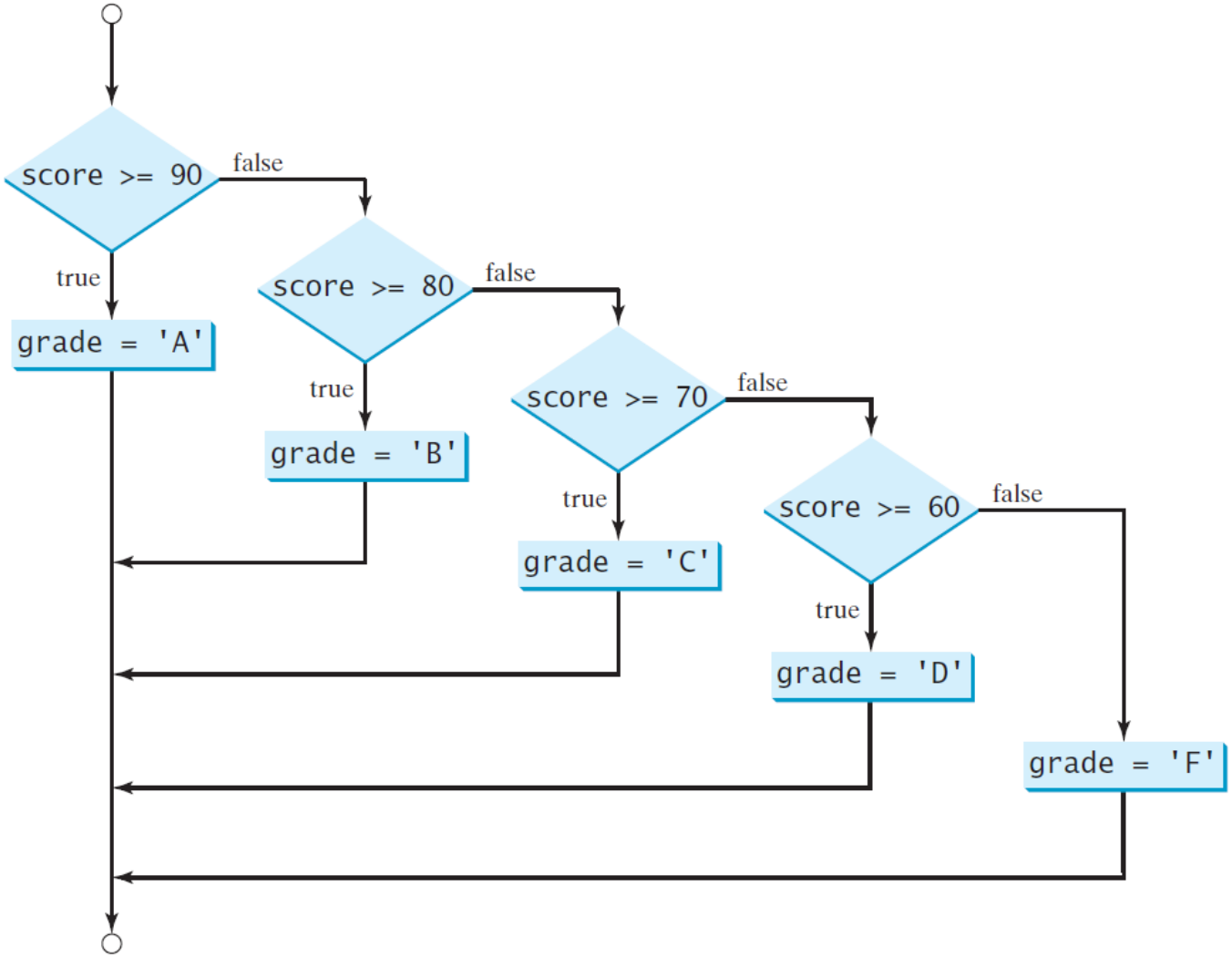
This is better

```
1  if score >= 90.0:
2      grade = 'A'
3  elif score >= 80.0:
4      grade = 'B'
5  elif score >= 70.0:
6      grade = 'C'
7  elif score >= 60.0:
8      grade = 'D'
9  else:
10     grade = 'F'
```

(b)

- While (a) works, the preferred format for multiple alternatives is shown in (b) using a multi-way if-elif-else statement.
- This multi-way if-elif-else style avoids deep indentation and makes the program easier to read.

Nested if and Multi-Way if-elif-else Statements



Trace if-elif-else Statement

Suppose score is **70.0**

```
1  if score >= 90.0:
2      grade = 'A'
3  elif score >= 80.0:
4      grade = 'B'
5  elif score >= 70.0:
6      grade = 'C'
7  elif score >= 60.0:
8      grade = 'D'
9  else:
10     grade = 'F'
11
```

The condition is False

Trace if-elif-else Statement

Suppose score is **70.0**

```
1  if score >= 90.0:
2      grade = 'A'
3  elif score >= 80.0:
4      grade = 'B'
5  elif score >= 70.0:
6      grade = 'C'
7  elif score >= 60.0:
8      grade = 'D'
9  else:
10     grade = 'F'
11
```

The condition is False

Trace if-elif-else Statement

Suppose score is **70.0**

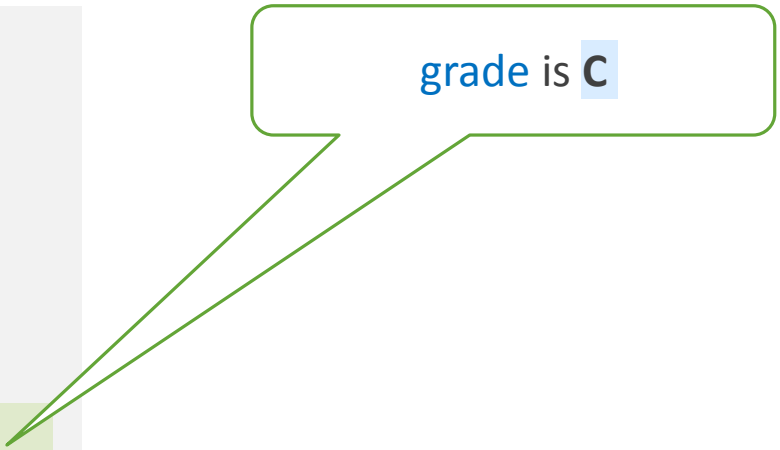
```
1  if score >= 90.0:
2      grade = 'A'
3  elif score >= 80.0:
4      grade = 'B'
5  elif score >= 70.0:
6      grade = 'C'
7  elif score >= 60.0:
8      grade = 'D'
9  else:
10     grade = 'F'
11
```

The condition is True

Trace if-elif-else Statement

Suppose score is **70.0**

```
1  if score >= 90.0:
2      grade = 'A'
3  elif score >= 80.0:
4      grade = 'B'
5  elif score >= 70.0:
6      grade = 'C'
7  elif score >= 60.0:
8      grade = 'D'
9  else:
10     grade = 'F'
11
```



Trace if-elif-else Statement

Suppose score is **70.0**

```
1  if score >= 90.0:
2      grade = 'A'
3  elif score >= 80.0:
4      grade = 'B'
5  elif score >= 70.0:
6      grade = 'C'
7  elif score >= 60.0:
8      grade = 'D'
9  else:
10     grade = 'F'
11
```

Exit the if statement

Note:
A condition is only tested when all the conditions that come before it are False.

Chinese Zodiac
Program 4

Write a program that will determine the Chinese Zodiac for a given year. Specifically, your program should prompt the user to enter a year and then determine the Zodiac and display the results.

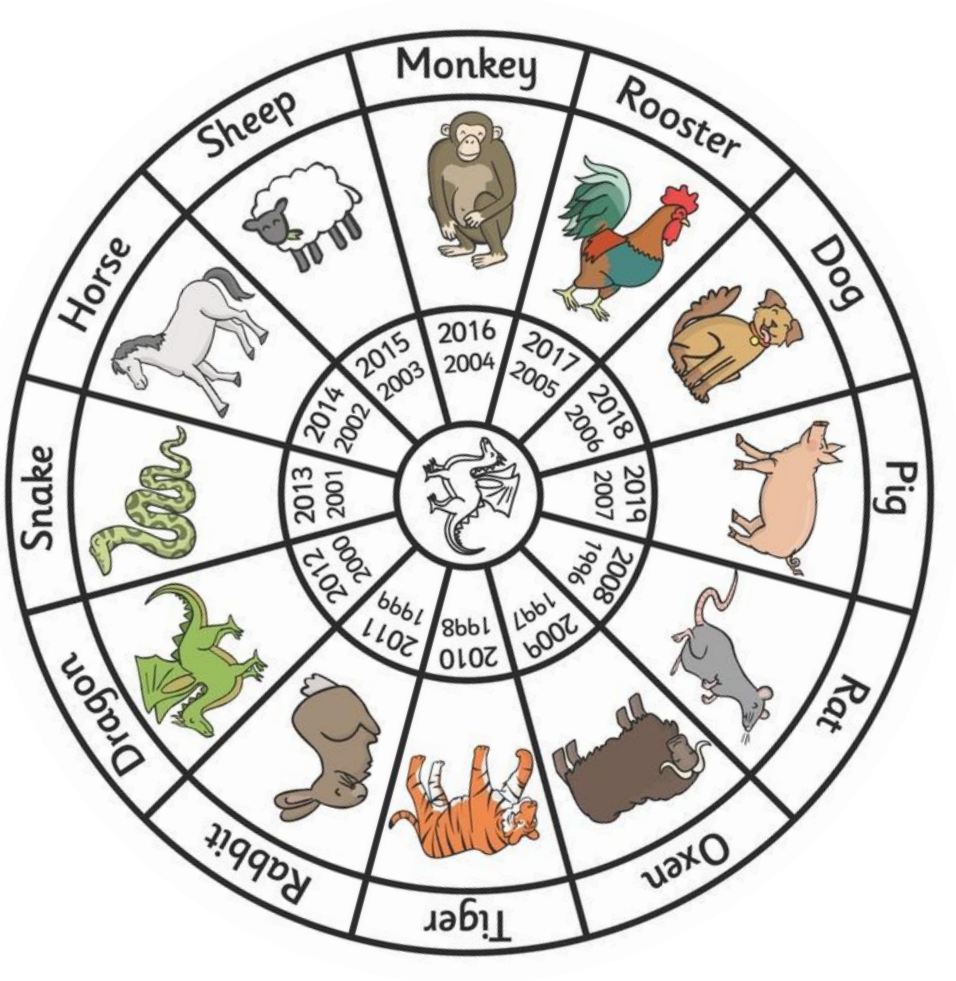
The Chinese zodiac sign is based on a 12-year cycle, and each year in this cycle is represented by an animal: monkey, rooster, dog, pig, rat, ox, tiger, rabbit, dragon, snake, horse, and sheep.

```
Enter a year: 1963  <Enter>
rabbit
```

```
Enter a year: 1877  <Enter>
ox
```

Chinese Zodiac
Phase 1: Problem-solving

- Zodiac is shown by graph below:

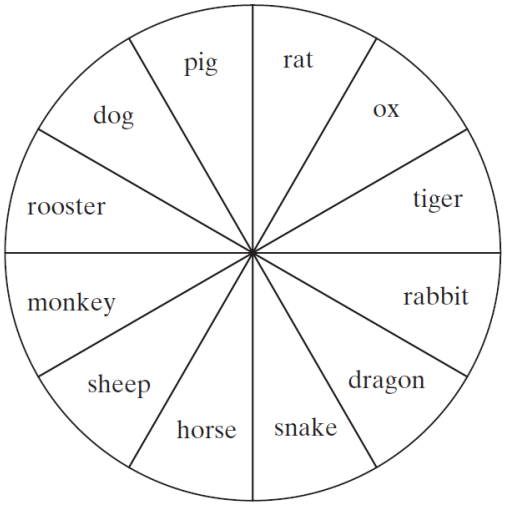


year % 12 =
- 0: monkey
- 1: rooster
- 2: dog
- 3: pig
- 4: rat
- 5: ox
- 6: tiger
- 7: rabbit
- 8: dragon
- 9: snake
- 10: horse
- 11: sheep

Chinese Zodiac
Phase 1: Problem-solving

Design your algorithm:

1. Ask the user to enter the year
2. Determine the correct Zodiac year
   - **zodiacYear = year % 12**



year % 12 =
0: monkey
1: rooster
2: dog
3: pig
4: rat
5: ox
6: tiger
7: rabbit
8: dragon
9: snake
10: horse
11: sheep

3. Print the result (zodiacYear)

Chinese Zodiac
Phase 2: Implementation

LISTING 4.5 ChineseZodiac.py

```
1  year = eval(input("Enter a year: "))
2  zodiacYear = year % 12
3  if zodiacYear == 0:
4      print("monkey")
5  elif zodiacYear == 1:
6      print("rooster")
7  elif zodiacYear == 2:
8      print("dog")
9  elif zodiacYear == 3:
10     print("pig")
11 elif zodiacYear == 4:
12     print("rat")
13 elif zodiacYear == 5:
14     print("ox")
15 elif zodiacYear == 6:
16     print("tiger")
17 elif zodiacYear == 7:
18     print("rabbit")
19 elif zodiacYear == 8:
20     print("dragon")
21 elif zodiacYear == 9:
22     print("snake")
23 elif zodiacYear == 10:
24     print("horse")
25 else:
26     print("sheep")
```

Run

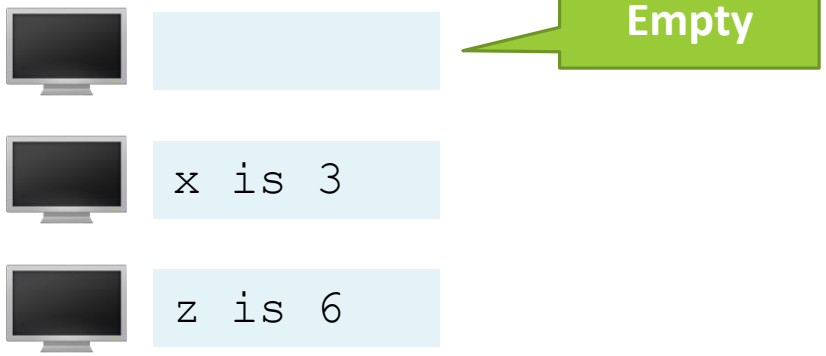
```
Enter a year: 1963 <Enter>
rabbit
```

```
Enter a year: 1877 <Enter>
ox
```

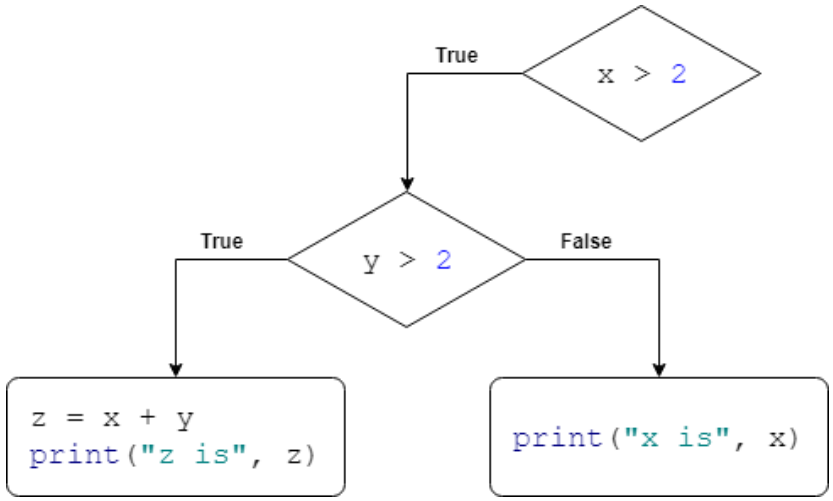
Check Point #8

Given the following code, show the output when:

- x = **2** and y = **3**
- x = **3** and y = **2**
- x = **3** and y = **3**



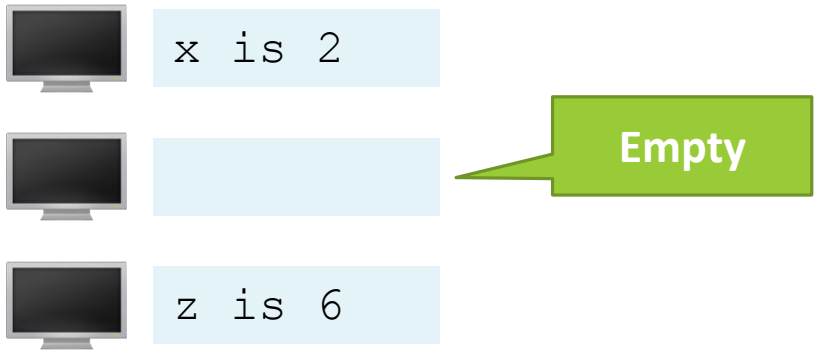

```
1  if x > 2:
2      if y > 2:
3          z = x + y
4          print("z is", z)
5      else:
6          print("x is", x)
```



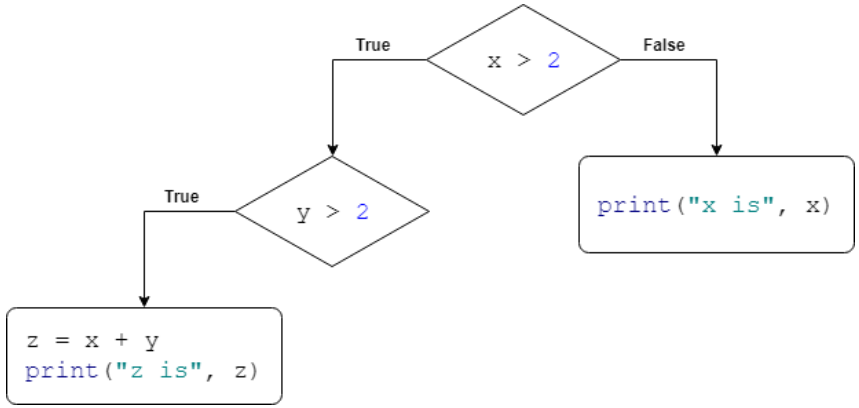
Check Point #9

Given the following code, show the output when:

- x = **2** and y = **4**
- x = **3** and y = **2**
- x = **3** and y = **3**



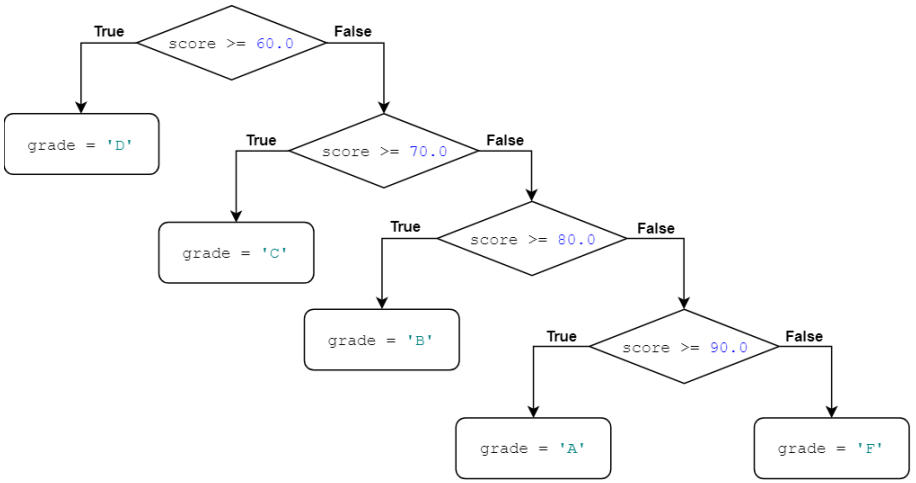

```
1  if x > 2:
2      if y > 2:
3          z = x + y
4          print("z is", z)
5  else:
6      print("x is", x)
```



Check Point #10

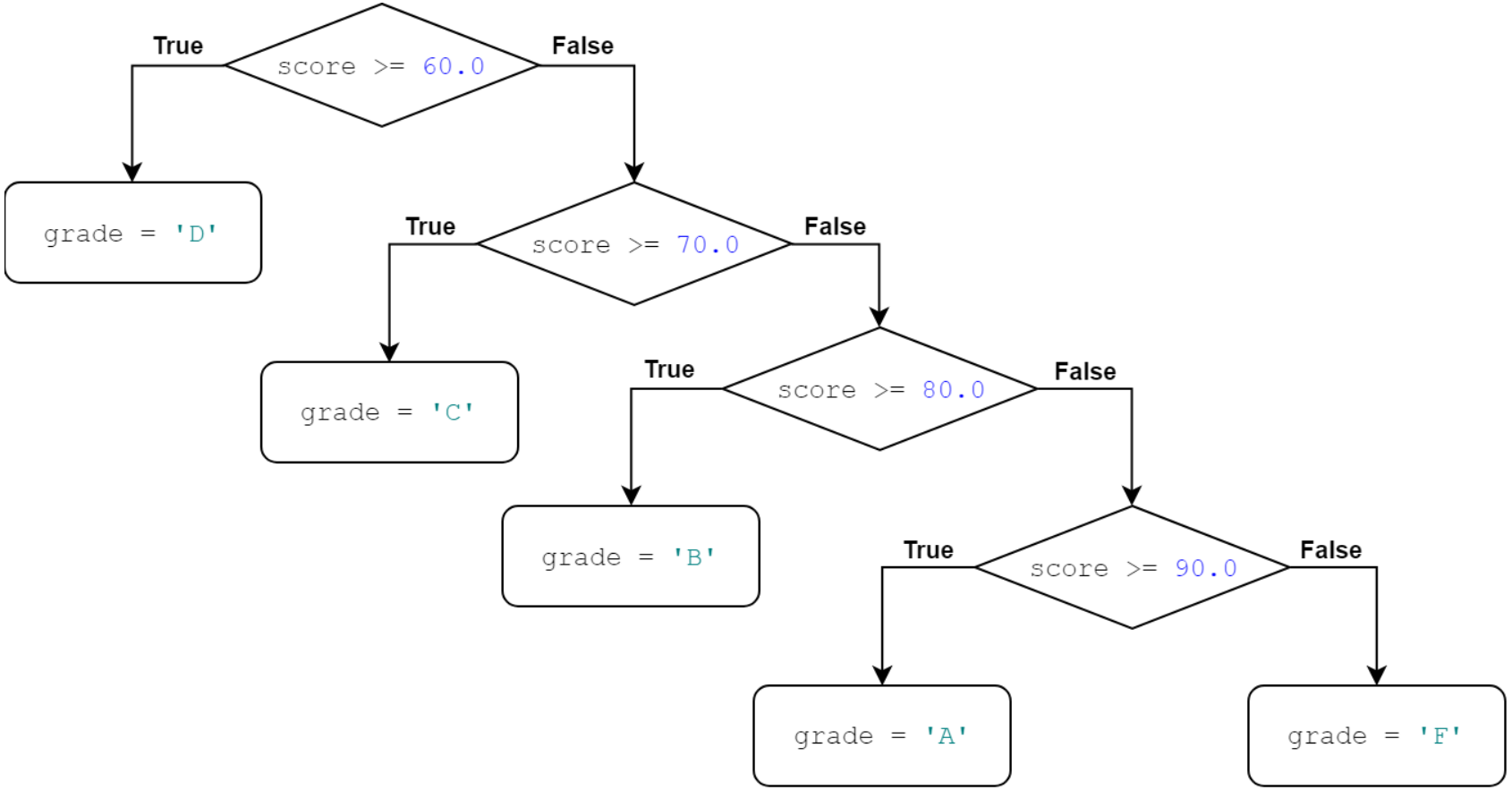
What is wrong in the following code?

```
1   if score >= 60.0:
2       grade = 'D'
3   elif score >= 70.0:
4       grade = 'C'
5   elif score >= 80.0:
6       grade = 'B'
7   elif score >= 90.0:
8       grade = 'A'
9   else:
10      grade = 'F'
```



- The code has a logic error. It will assign "D" always when score is equal or greater than **60**. It will not, for example, assign "A" if the score is equal or greater than **90**.
- This is because a condition is only tested when all the conditions that come before it are False.

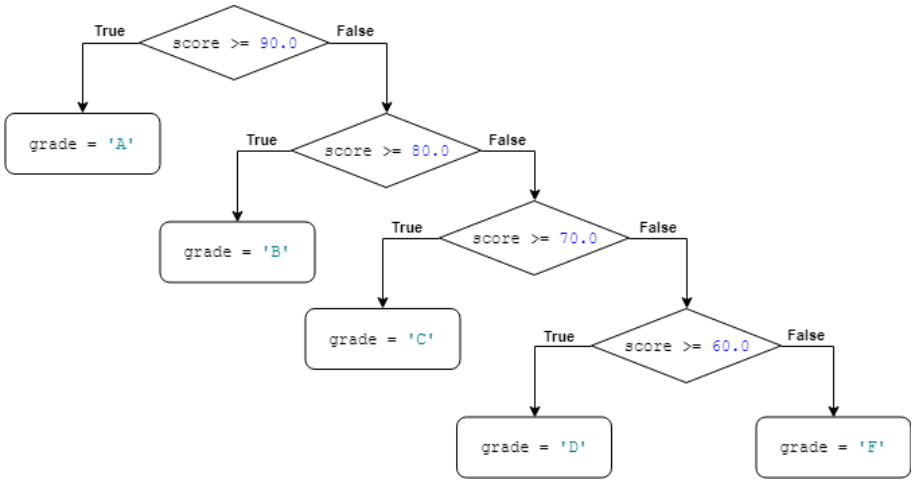
Check Point #10



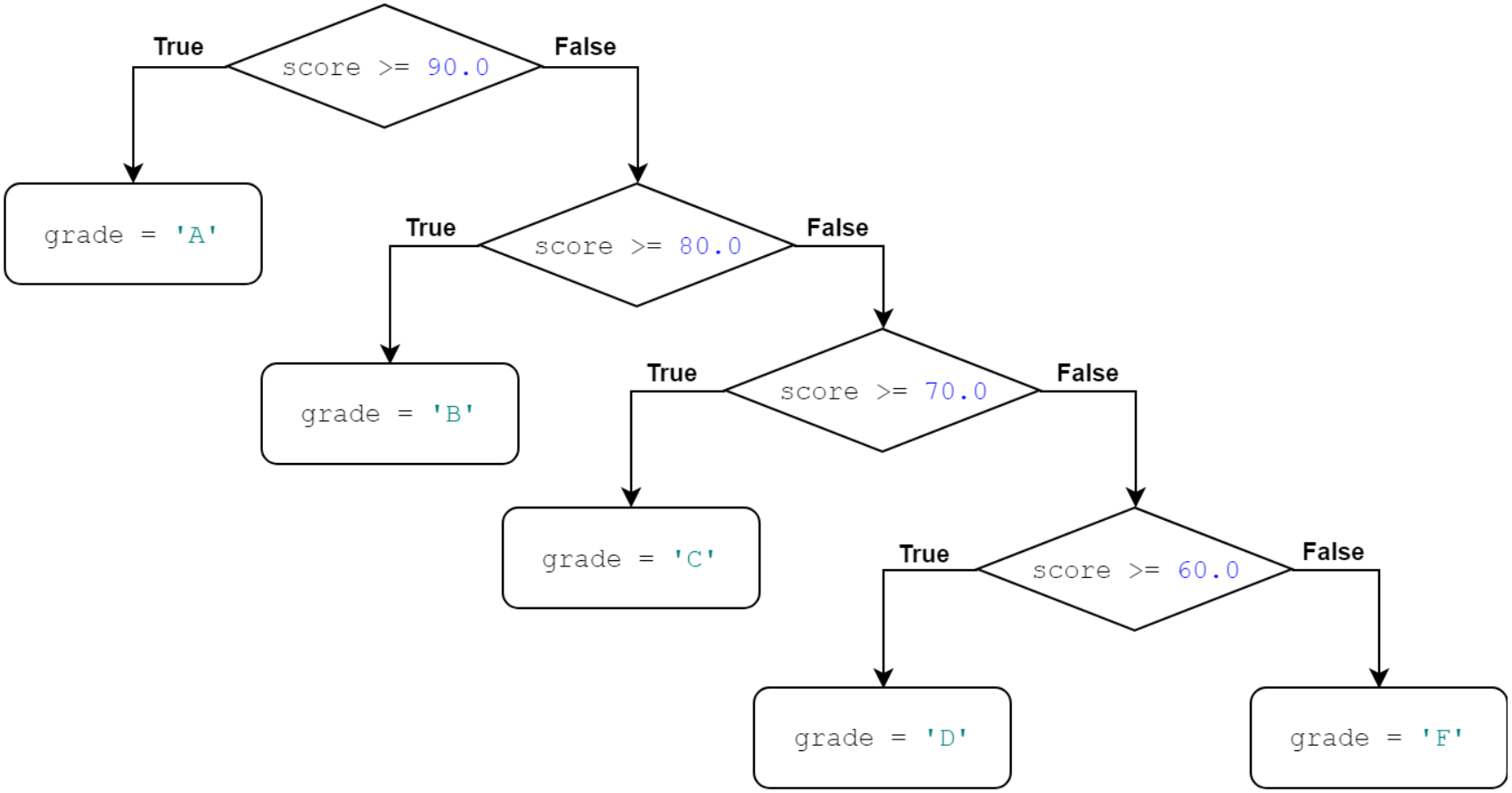
Check Point #10

- This is the fix of the previous code:

```
1   if score >= 90.0:
2       grade = 'A'
3   elif score >= 80.0:
4       grade = 'B'
5   elif score >= 70.0:
6       grade = 'C'
7   elif score >= 60.0:
8       grade = 'D'
9   else:
10      grade = 'F'
```



Check Point #10

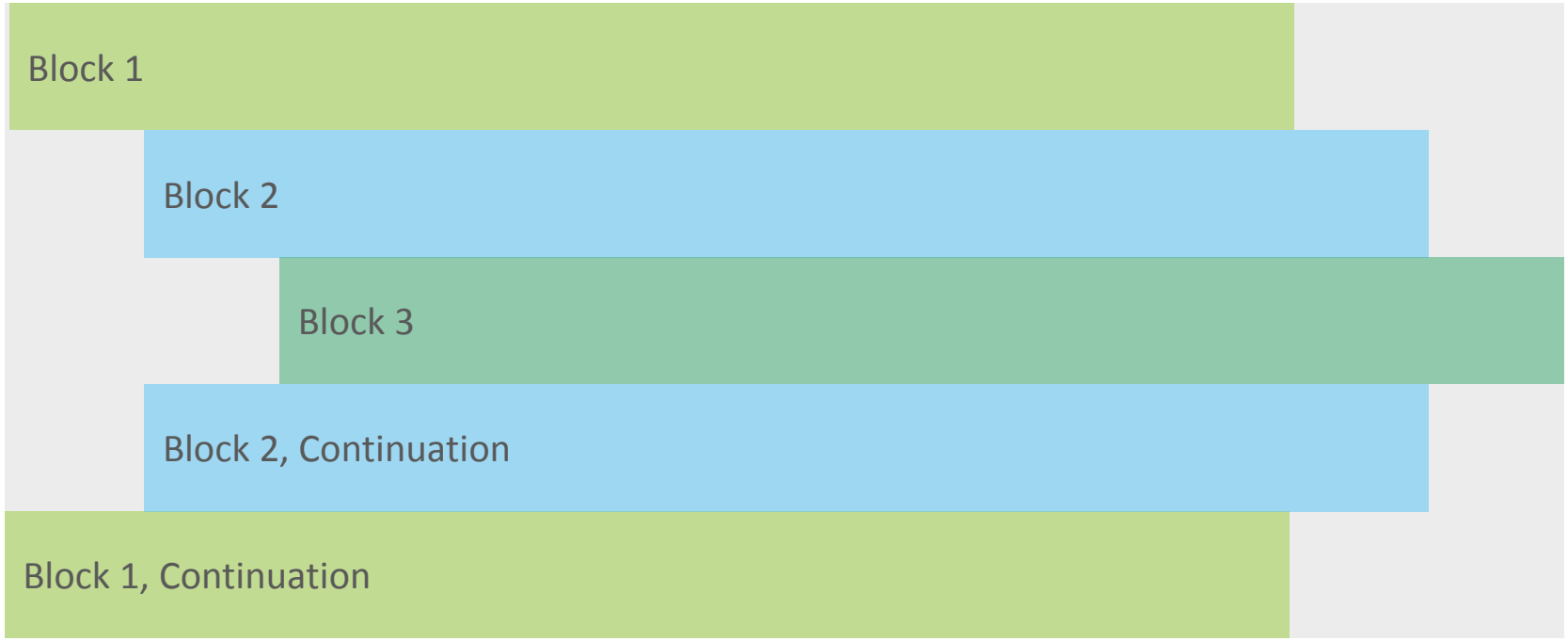


4.8. Common Errors in Selection Statements

- Common Errors
- Common Pitfalls
- Check Point #11 - 13

Common Errors

- Most common errors in selection statements are caused by incorrect indentation.



Common Errors
Example 1

- Consider the following code in (a) and (b):

```
1  import math
2  radius = -20
3  if radius >= 0:
4      area = radius * radius * math.pi
5  print("The area is", area)
```

```
  print("The area is", area)
NameError: name 'area' is not defined
```

(a)

```
1  import math
2  radius = -20
3  if radius >= 0:
4      area = radius * radius * math.pi
5      print("The area is", area)
```

Empty

(b)

- In (a), the print statement is not in the if block.
- To place it in the if block, you have to indent it, as shown in (b).
- By the way, (a) has a runtime error: NameError: name 'area' is not defined.

Common Errors
Example 2

- Consider the following code in (a) and (b):

```
1  i = 1
2  j = 2
3  k = 3
4  if i > j:
5      if i > k:
6          print('A')
7  else:
8      print('B')
```

```
B
```

(a)

```
1  i = 1
2  j = 2
3  k = 3
4  if i > j:
5      if i > k:
6          print('A')
7      else:
8          print('B')
```

Empty

(b)

  - The code in (a) has two if clauses and one else clause. **Which if clause is matched by the else clause?**
  - The indentation indicates that the else clause matches the first if clause in (a) and the second if clause in (b).

Common Pitfalls

- Common Pitfall 1:
  - Testing equality of double values.

- Common Pitfall 2:
  - Duplicated statements in if-else or if-elif-else statements.

Common Pitfalls
Pitfall 1

- Simplify Boolean variable assignment.
  - Often, new programmers write code like (a).

```
1  number = eval(input("Enter number: "))
2  if number % 2 == 0:
3      even = True
4  else:
5      even = False
```

(a)

```
1  number = eval(input("Enter number: "))
2  even = number % 2 == 0
```

(b)

  - This is not an error.
  - But it is better written (and shorter) as shown in (b).
  - (b) is equivalent to (a).

Common Pitfalls
Pitfall 2

- Avoid duplicating code in different cases.
  - Often, new programmers write duplicate code that should be combined in one place.

```
1  total = eval(input("Enter total: "))
2
3  if (total >= 100):
4      discount = 10 / 100
5      total = total - (discount * total)
6      print("Final Total: ", total)
7  else:
8      discount = 5 / 100
9      total = total - (discount * total)
10     print("Final Total: ", total)
```

(a)

```
1  total = eval(input("Enter total: "))
2  discount = 0
3
4  if (total >= 100):
5      discount = 10 / 100
6  else:
7      discount = 5 / 100
8
9  total = total - (discount * total)
10 print("Final Total: ", total)
```

(b)

  - This is not an error.
  - But the new code (b) removes the duplication and makes the code easy to maintain, because you only need to change in one place if the print statement is modified.

Check Point #11

Rewrite the following statement using a Boolean expression:

```
1 if count % 10 == 0:
2     newLine = True
3 else:
4     newLine = False
```

➢ Solution:

```
1 newLine = count % 10 == 0
```

Check Point #12

Are the following statements correct? Which one is better?

```
1  if age < 16:
2      print("Cannot get a driver's license")
3  if age >= 16:
4      print("Can get a driver's license")
```

(a)

```
1  if age < 16:
2      print("Cannot get a driver's license")
3  else:
4      print("Can get a driver's license")
```

(b)

- Yes, they are correct.
- (b) is better than (a) because it is concise and easy to read.

Check Point #13

What is the output of the following code if number is **14**, **15**, and **30**?

```
1  if number % 2 == 0:
2      print(number, "is even")
3  if number % 5 == 0:
4      print(number, "is multiple of 5")
```

(a)

```
14 is even
```

```
15 is multiple of 5
```

```
30 is even
30 is multiple of 5
```

```
1  if number % 2 == 0:
2      print(number, "is even")
3  elif number % 5 == 0:
4      print(number, "is multiple of 5")
```

(b)

```
14 is even
```

```
15 is multiple of 5
```

```
30 is even
```

4.9. Case Study: Computing Body Mass Index

- Program 5: Computing BMI

Computing BMI
Program 5

Write a program that computes the Body Mass Index (BMI) for the user. Your program should prompt the user to enter a weight in pounds and height in inches. Your program should then compute and display the BMI and its interpretation for the user.

BMI	Interpretation
Below 18.5	Underweight
18.5-24.9	Normal
25.0-29.9	Overweight
Above 30.0	Obese

$$\text{BMI} = \frac{\text{Weight (kg)}}{\text{Height (m)}^2}$$

$$1\ pound = 0.45359237\ kilograms$$

$$1\ inch = 0.0254\ meters$$

```
Enter weight in pounds: 146  <Enter>
Enter height in inches: 70  <Enter>
BMI is 20.95
Normal
```

Computing BMI
Phase 1: Problem-solving

- BMI is a measure of health based on the height and weight.
- BMI is calculated by taking the weight (in kilograms) and then dividing it by the square of the height (in meters)

$$\text{BMI} = \frac{\text{Weight (kg)}}{\text{Height (m)}^2} = \frac{\text{Weight (kg)}}{\text{Height (m)} \times \text{Height (m)}}$$

- The interpretation of BMI for people 20 years or older is as follows:

BMI	Interpretation
Below 18.5	Underweight
18.5-24.9	Normal
25.0-29.9	Overweight
Above 30.0	Obese

Computing BMI
Phase 1: Problem-solving

- So the user input is in pounds and inches
- The BMI equation is in kilograms and meters
- Therefore, you will need to convert from:
  - pounds to kilograms
    - One pound is 0.45359237 kilograms
    - $Weight\ (kg) = 0.45359237 \times Weight\ (pound)$
  - inches to meters
    - one inch is 0.0254 meters
    - $Height\ (m) = 0.0254 \times Height\ (inch)$

Computing BMI
Phase 1: Problem-solving

Design your algorithm:

1. Ask the user to enter the weight and height
2. Convert weight in pounds to kilograms
   - $weightInKilograms = weight * 0.45359237$
3. Convert height in inches to meters
   - $heightInMeters = height * 0.0254$
4. Compute BMI using BMI equation
   - bmi = weightInKilograms / (heightInMeters * heightInMeters)
5. Print the result (bmi)
   - Print the interpretation as the following:
     - "Underweight" if bmi < 18.5
     - "Normal" if bmi < 25
     - "Overweight" if bmi < 30
     - "Obese" if bmi >= 30

Computing BMI
Phase 2: Implementation

LISTING 4.6 ComputeBMI.py

```
1  # Prompt the user to enter weight in pounds
2  weight = eval(input("Enter weight in pounds: "))
3
4  # Prompt the user to enter height in inches
5  height = eval(input("Enter height in inches: "))
6
7  KILOGRAMS_PER_POUND = 0.45359237  # Constant
8  METERS_PER_INCH = 0.0254  # Constant
9
10 # Compute BMI
11 weightInKilograms = weight * KILOGRAMS_PER_POUND
12 heightInMeters = height * METERS_PER_INCH
13 bmi = weightInKilograms / (heightInMeters * heightInMeters)
14
15 # Display result
16 print("BMI is", format(bmi, ".2f"))
17 if bmi < 18.5:
18     print("Underweight")
19 elif bmi < 25:
20     print("Normal")
21 elif bmi < 30:
22     print("Overweight")
23 else:
24     print("Obese")
```

Computing BMI
Example Runs of The Program

```
Enter weight in pounds: 146  <Enter>
Enter height in inches: 70  <Enter>
BMI is 20.95
Normal
```

```
Enter weight in pounds: 176  <Enter>
Enter height in inches: 66  <Enter>
BMI is 28.41
Overweight
```

Computing BMI
Trace The Program Execution

```
Enter weight in pounds: 146   <Enter>
Enter height in inches: 70   <Enter>
BMI is 20.95
Normal
```

line#	weight	height	weightInKilograms	heightInMeters	bmi	output
2	146					
5		70				
11			66.22448602			
12				1.778		
13					20.9486	
16						BMI is 20.95
20						Normal

Computing BMI
Discussion

- The two named constants, KILOGRAMS_PER_POUND and METERS_PER_INCH, are defined in lines 7-8.
- Named constants were introduced in Chapter 2.
- Using named constants here makes programs easy to read.
- Unfortunately, there is no special syntax for defining named constants in Python.
- Named constants are treated just like variables in Python.
- This book uses the format of writing constants in all uppercase letters to distinguish them from variables and separates the words in constants with an underscore (_).

4.11. Logical Operators

- Truth Table for Operator not
- Truth Table for Operator and
- Truth Table for Operator or
- Program 6: Test Boolean Operators
- De Morgan's law
- Notes
- Check Point #14 - #21

Logical Operators

- We have used conditional statements to help us determine if the execution should take one path (true path) or another path (false path).
- But until now, these conditional statements have been very basic.
- Usually, whether a statement is executed is determined by a **combination of several conditions**.
- You can use logical operators to combine these conditions to form a compound Boolean expression.

Logical Operators

- Logical operators, also known as Boolean operators, operate on Boolean values to create a new Boolean value.
- The following slide shows the three logical operators we will use.
- The following slides show a truth table for each logical operator and some examples.

Logical Operators

TABLE 4.3 Boolean Operators

Operator	*Description*
`not`	logical negation
`and`	logical conjunction
`or`	logical disjunction

Truth Table for Operator not

TABLE 4.4 Truth Table for Operator `not`

p	not p	*Example (assume* `age = 24, gender = 'F'`*)*
`True`	`False`	`not (age > 18)` is `False`, because `(age > 18)` is `True`.
`False`	`True`	`not (gender == 'M')` is `True`, because `(gender == 'M')` is `False`.

Truth Table for Operator not
Examples


```
>>> not True
False
>>> not ( 20 > 60 )
True
>>> not 60 > 20
False
>>> not ( not ( True ) )
True
>>> not not False
False
```

Truth Table for Operator and

TABLE 4.5 Truth Table for Operator `and`

p_1	p_2	p_1 `and` p_2	*Example (assume* `age = 24, gender = 'F'`*)*
`False`	`False`	`False`	`(age > 18) and (gender == 'F')` is `True`, because `(age > 18)` and `(gender == 'F')` are both `True`.
`False`	`True`	`False`	
`True`	`False`	`False`	`(age > 18) and (gender != 'F')` is `False`, because `(gender != 'F')` is `False`.
`True`	`True`	`True`	

Truth Table for Operator and Examples


```
>>> True and False and True and True
False
>>> True and not False
True
>>> not not True and 10 > 20
False
>>> not False and not False and not not not False
True
>>> not (True and False) and True
True
>>> not True and False and True
False
```

Truth Table for Operator or

TABLE 4.6 Truth Table for Operator `or`

p_1	p_2	p_1 or p_2	*Example (assume* `age = 24, gender = 'F'`*)*
`False`	`False`	`False`	`(age > 34) or (gender == 'F')` is `True`, because `(gender == 'F')` is `True`.
`False`	`True`	`True`	
`True`	`False`	`True`	`(age > 34) or (gender == 'M')` is `False`, because `(age > 34)` and `(gender == 'M')` are both `False`.
`True`	`True`	`True`	

Truth Table for Operator or
Examples


```
>>> True or False
True
>>> True or False and False and False and True
True
>>> False or not True
False
>>> not (True and False) or 20 > 10
True
>>> not True and False or True
True
>>> not True and (False or True)
False
```

Test Boolean Operators
Program 6

Write a program that tests the usage of Boolean operators. Specifically, your program should prompt the user to enter one integer. Your program should then determine if the value is divisible by **2** and **3**, by **2** or **3**, or by **2** or **3** but not both.

```
Enter an integer: 18  <Enter>
18 is divisible by 2 and 3
18 is divisible by 2 or 3
```

```
Enter an integer: 15  <Enter>
15 is divisible by 2 or 3
15 is divisible by 2 or 3, but not both
```

Test Boolean Operators
Phase 1: Problem-solving

- Note:
  - So how do we check for divisibility?
  - We use mod (%).
  - Example: check if some number, x, is divisible by **3**

```
if x % 3 == 0
```

  - This says: if we divide x by **3** and the remainder is zero …
  - And that is exactly what we want!
  - However, we must check the divisibility of two numbers
    - both **2** and **3**
  - This means we must use logical operators

```
if x % 2 == 0 and x % 3 == 0
```

Test Boolean Operators
Phase 1: Problem-solving

Design your algorithm:

1. Ask the user to enter the number
2. If (number % 2 == 0) **and** (number % 3 == 0)
   - Print: number is divisible by 2 and 3
3. If (number % 2 == 0) **or** (number % 3 == 0)
   - Print: number is divisible by 2 or 3
4. If ((number % 2 == 0) **or** (number % 3 == 0))
   and (**not** ((number % 2 == 0) **and** (number % 3 == 0)))
   - Print: number is divisible by 2 or 3, but not both

Test Boolean Operators
Phase 2: Implementation

LISTING 4.8 TestBooleanOperators.py

```
1  # Receive an input
2  number = eval(input("Enter an integer: "))
3
4  if number % 2 == 0 and number % 3 == 0:
5      print(number, "is divisible by 2 and 3")
6
7  if number % 2 == 0 or number % 3 == 0:
8      print(number, "is divisible by 2 or 3")
9
10 if (number % 2 == 0 or number % 3 == 0) and \
11        not (number % 2 == 0 and number % 3 == 0):
12     print(number, "is divisible by 2 or 3, but not both")
13
```

Run

```
Enter an integer: 18  <Enter>
18 is divisible by 2 and 3
18 is divisible by 2 or 3
```

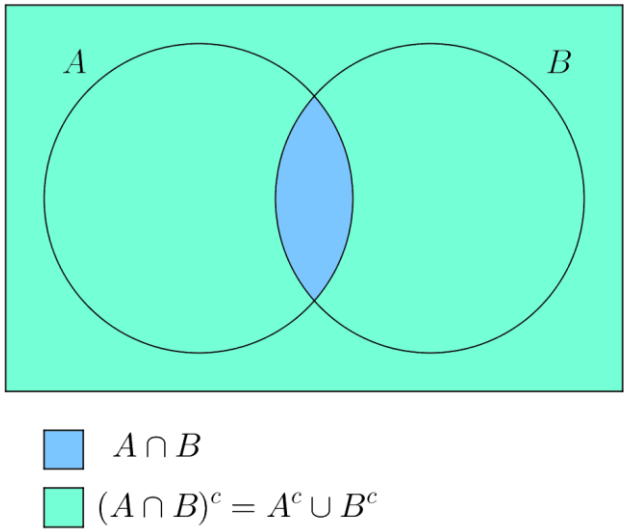
```
Enter an integer: 15  <Enter>
15 is divisible by 2 or 3
15 is divisible by 2 or 3, but not both
```

De Morgan's law (1)

```
not (condition1 and condition2)
```

is the same as

```
not condition1 or not condition2
```



- Example, the following Boolean expression:

```
not (number % 2 == 0 and number % 3 == 0)
```

is better written as:

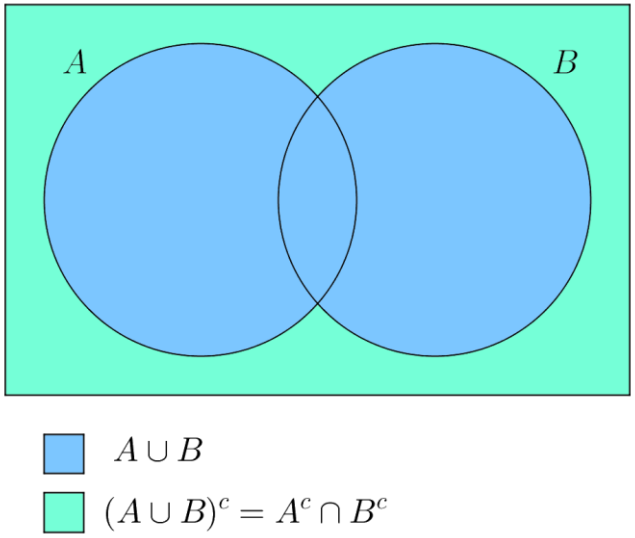
```
number % 2 != 0 or number % 3 != 0
```

De Morgan's law (2)

```
not (condition1 or condition2)
```

is the same as

```
not condition1 and not condition2
```



- Example, the following Boolean expression:

```
not (number == 2 or number == 3)
```

is better written as:

```
number != 2 and number != 3
```

Notes

- If one of the operands of an and operator is False, the expression is False.
  - Example: when evaluating p1 and p2
    - Python first evaluates p1
    - if p1 is True, evaluates p2
    - if p1 is False, it does not evaluate p2
- if one of the operands of an or operator is True, the expression is True.
  - Example: when evaluating p1 or p2
    - Python first evaluates p1
    - if p1 is False, evaluates p2
    - if p1 is True, it does not evaluate p2
- Python uses these properties to improve the performance of these operators.

Notes

- The following Boolean expression:

```
number >= x and number < y
```

  can be simplified by using an equivalent expression:

```
x <= number < y
```

- The following Boolean expression:

```
number != x and number == y
```

  can be simplified by using an equivalent expression:

```
x != number == y
```

Check Point #14

Assuming that x is **1**, show the result of the following Boolean expressions:

1.	`True and (3 > 4)`	`False`
2.	`not (x > 0) and (x > 0)`	`False`
3.	`(x > 0) or (x < 0)`	`True`
4.	`(x != 0) or (x == 0)`	`True`
5.	`(x >= 0) or (x < 0)`	`True`
6.	`(x != 1) == (not (x == 1))`	`True`

Check Point #15

Write a Boolean expression that evaluates to True if variable num is between **1** and **100**.

- Solution:

```
1 <= num <= 100
```

Or (equivalent):

```
num >= 1 and num <= 100
```

Check Point #16

Write a Boolean expression that evaluates to True if variable num is between **1** and **100** **or** num is negative.

- Solution:

```
(1 <= num <= 100) or (num < 0)
```

Or (equivalent):

```
(num >= 1 and num <= 100) or (num < 0)
```

Check Point #17

Assuming x = **4** and y = **5**, show the result of the following Boolean expressions:

1. `x >= y >= 0` — `False`
2. `x <= y >= 0` — `True`
3. `x != y == 5` — `True`
4. `(x != 0) or (x == 0)` — `True`

Check Point #18

Write a Boolean expression that evaluates to True if age is greater than **13** and less than **18**.

- Solution:

```
13 < age < 18
```

Or (equivalent):

```
age > 13 and age < 18
```

Check Point #19

Write a Boolean expression that evaluates to True if weight is greater than **50 or** height is greater than **160**.

- Solution:

```
weight > 50 or height > 160
```

Check Point #20

Write a Boolean expression that evaluates to True if weight is greater than **50 and** height is greater than **160**.

- Solution:

```
weight > 50 and height > 160
```

Check Point #21

Write a Boolean expression that evaluates to True if **either** weight is greater than **50 or** height is greater than **160**, but not both.

- Solution:

```
(weight > 50 or height > 160) and not (weight > 50 and height > 160)
```

4.12. Case Study: Determining Leap Years

- Program 7: Leap Year

Leap Year
Program 7

Write a program that determines if a given year is a leap year. Specifically, ask the user to enter a year. Then determine if that year is a leap year and display the results.

Note: A year is a leap year if it is divisible by **4** but not by **100** **or** if it is divisible by **400**.

```
Enter a year: 2008  <Enter>
2008 is a leap year? True
```

```
Enter a year: 1900  <Enter>
1900 is a leap year? False
```

```
Enter a year: 2002  <Enter>
2002 is a leap year? False
```

Leap Year
Phase 1: Problem-solving

- What is a leap year?
  - A leap year has 366 days (instead of 365)
  - Why?
  - The earth takes approximately 365.25 days to circle around the sun
  - However, the Gregorian year has only 365 days
  - Therefore, every four years, the number of days is increased to 366
- A year is a leap year if it is divisible by **4** but not by **100** **or** if it is divisible by **400**.

Leap Year
Phase 1: Problem-solving

- Which years are leap years?
- There are three criteria:
  1. A leap year is divisible by **4**

```
isLeapYear = (year % 4 == 0)
```

  2. A leap year is divisible by **4 but not** by **100**

```
isLeapYear = isLeapYear and (year % 100 != 0)
```

  3. A leap year is divisible by **4 but not** by **100 or** divisible by **400**

```
isLeapYear = isLeapYear or (year % 400 == 0)
```

Leap Year
Phase 1: Problem-solving

- So, you can use the following Boolean expressions to determine whether a year is a leap year:

```
# A leap year is divisible by 4
isLeapYear = (year % 4 == 0)
# A leap year is divisible by 4 but not by 100
isLeapYear = isLeapYear and (year % 100 != 0)
# A leap year is divisible by 4 but not by 100 or divisible by 400
isLeapYear = isLeapYear or (year % 400 == 0)
```

- or you can combine all these expressions into one, like this:

```
isLeapYear = (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
```

Leap Year
Phase 1: Problem-solving

Design your algorithm:

1. Ask the user to enter the year
2. If (year % 4 == 0 **and** year % 100 **!=** 0) **or** (year % 400 == 0)
   - Print: year is a leap year? True
3. Otherwise
   - Print: year is a leap year? False

Leap Year
Phase 2: Implementation

LISTING 4.9 LeapYear.py

```
1  year = eval(input("Enter a year: "))
2
3  # Check if the year is a leap year
4  isLeapYear = (year % 4 == 0 and year % 100 != 0) or \
5     (year % 400 == 0)
6
7  # Display the result
8  print(year, "is a leap year?", isLeapYear)
9
```

Run

```
Enter a year: 2008  <Enter>
2008 is a leap year? True
```

```
Enter a year: 1900  <Enter>
1900 is a leap year? False
```

```
Enter a year: 2002  <Enter>
2002 is a leap year? False
```

4.13. Case Study: Lottery

- Program 8: Lottery

Lottery
Program 8

Write a program to play a lottery. The program randomly generates a two-digit number, prompts the user to enter a two-digit number, and determines whether the user wins according to the following rules:

1. If the user's input matches the lottery in the exact order, the award is $10,000.
2. If all the digits in the user's input match all the digits in the lottery number, the award is $3,000.
3. If one digit in the user's input matches a digit in the lottery number, the award is $1,000.

```
Enter your lottery pick (two digits): 45  <Enter>
The lottery number is 12
Sorry, no match
```

```
Enter your lottery pick (two digits): 23  <Enter>
The lottery number is 34
Match one digit: you win $1,000
```

Lottery
Phase 1: Problem-solving

- So how do we compare digits?
  - Give a two-digit number, how can isolate the individual digits in order to compare them?
  - Example: given the number **73**, how can we extract the **7** and the **3**? How can we "get" them as individual numbers?
- Solution: integer division (//) and mod (%)
  - Example: **73**
  - 73 // 10 = **7**
  - 73 % 10 = **3**
- This is exactly what we want!

Note:
you will use mod (%) a lot in this course!

Lottery
Phase 1: Problem-solving

Design your algorithm:

1. Randomly generate a lottery number between 10 and 99.
   - lottery = random.randint(10, 99)
2. Ask the user to enter the two-digit number (guess)
3. Get digits from lottery
   - lotteryDigit1 = lottery // 10
   - lotteryDigit2 = lottery % 10
4. Get digits from guess
   - guessDigit1 = guess // 10
   - guessDigit2 = guess % 10

Lottery
Phase 1: Problem-solving

Design your algorithm:

5. Compare user number (guess) with winning number (lottery) and determine winning amount (if any).
   - First check whether the guess matches the lottery number exactly.
     - if guess == lottery
   - If not, check whether the reversal of the guess matches the lottery number.
     - elif (guessDigit2 == lotteryDigit1 and guessDigit1 == lotteryDigit2)
   - If not, check whether one digit is in the lottery number.
     - elif (guessDigit1 == lotteryDigit1 or guessDigit1 == lotteryDigit2 or guessDigit2 == lotteryDigit1 or guessDigit2 == lotteryDigit2)
   - If not, nothing matches and display Sorry, no match.
     - else
6. Display results to user

Lottery
Phase 2: Implementation

LISTING 4.10 Lottery.py

```
1  import random
2
3  # Generate a lottery
4  lottery = random.randint(10, 99)
5
6  # Prompt the user to enter a guess
7  guess = eval(input("Enter your lottery pick (two digits): "))
8
9  # Get digits from lottery
10 lotteryDigit1 = lottery // 10
11 lotteryDigit2 = lottery % 10
12
13 # Get digits from guess
14 guessDigit1 = guess // 10
15 guessDigit2 = guess % 10
16
17 print("The lottery number is", lottery)
```

Run

Lottery
Phase 2: Implementation

LISTING 4.10 Lottery.py

```
18
19  # Check the guess
20  if guess == lottery:
21      print("Exact match: you win $10,000")
22  elif (guessDigit2 == lotteryDigit1 and \
23    guessDigit1 == lotteryDigit2):
24      print("Match all digits: you win $3,000")
25  elif (guessDigit1 == lotteryDigit1
26          or guessDigit1 == lotteryDigit2
27          or guessDigit2 == lotteryDigit1
28          or guessDigit2 == lotteryDigit2):
29      print("Match one digit: you win $1,000")
30  else:
31      print("Sorry, no match")
```

Run

```
Enter your lottery pick (two digits): 45  <Enter>
The lottery number is 12
Sorry, no match
```

```
Enter your lottery pick (two digits): 23  <Enter>
The lottery number is 34
Match one digit: you win $1,000
```

Lottery
Trace The Program Execution

```
Enter your lottery pick (two digits): 23  <Enter>
The lottery number is 34
Match one digit: you win $1,000
```

variable \ line#	4	7	10	11	14	15	29
lottery	34						
guess		23					
lotteryDigit1			3				
lotteryDigit2				4			
guessDigit1					2		
guessDigit2						3	
output							Match one digit: you win $1,000

4.14. Conditional Expressions

- Check Point #22 - #27

Conditional Expressions

- You might want to assign a value to a variable that is restricted by certain conditions.
- For example, the following statement assigns **1** to y if x is greater than **0**, and **-1** to y if x is less than or equal to **0**.

```
if x > 0:
    y = 1
else:
    y = -1
```

- Alternatively, you can use a conditional expression to achieve the same result.

```
y = 1 if x > 0 else -1
```

Conditional Expressions

- A conditional expression evaluates an expression based on a condition.
- Conditional expressions are in a completely different style. The syntax is:

```
expression1 if boolean-expression else expression2
```

- The result of this conditional expression is expression1 if boolean-expression is True; otherwise, the result is expression2.

Conditional Expressions
Example 1

- Given two numbers, number1 and number2, save the larger into a variable called max.
- You can do this with an if/else statement

```
if number1 > number2:
    max = number1
else:
    max = number2
```

- Or you can use one conditional expression as follows:

```
max = number1 if number1 > number2 else number2
```

Conditional Expressions
Example 2

- Given a variable, number, display the message "number is even" if number is even; otherwise, display "number is odd".
- You can do this with an if/else statement

```
if number % 2 == 0:
    print(number, "is even")
else:
    print(number, "is odd")
```

- Or you can use one conditional expression as follows:

```
print(number, "is even" if number % 2 == 0 else "is odd")
```

Check Point #22

Suppose that when you run the following program, you enter the input **2**, **3**, **6** from the console. What is the output?

```
1 x, y, z = eval(input("Enter three numbers: "))
2 print("sorted" if x < y and y < z else "not sorted")
```

- The output is: sorted
- It is equivalent to:

```
1 x, y, z = eval(input("Enter three numbers: "))
2 if x < y and y < z:
3     print("sorted")
4 else:
5     print("not sorted")
```

Check Point #23

Rewrite the following if statement using a conditional expression:

```
if ages >= 16:
    ticketPrice = 20
else:
    ticketPrice = 10
```

- Solution:

```
ticketPrice = 20 if ages >= 16 else 10
```

Check Point #24

Rewrite the following if statement using a conditional expression:

```
if count % 10 == 0:
    print(count)
else:
    print(count, end = " ")
```

- Solution:

```
print(count, end = "\n" if count % 10 == 0 else " ")
```

Check Point #25

Rewrite the following conditional expressions using if/else statements:

```
score = 3 * scale if x > 10 else 4 * scale
```

- Solution:

```
if x > 10:
    score = 3 * scale
else:
    score = 4 * scale
```

Check Point #26

Rewrite the following conditional expressions using if/else statements:

```
tax = income * 0.2 if income > 10000 else income * 0.17 + 1000
```

- Solution:

```
if income > 10000:
    tax = income * 0.2
else:
    tax = income * 0.17 + 1000
```

Check Point #27

Rewrite the following conditional expressions using if/else statements:

```
print(i if number % 3 == 0 else j)
```

➢ Solution:

```
if number % 3 == 0:
    print(i)
else:
    print(j)
```

4.15. Operator Precedence and Associativity

- Operator Precedence
- Associativity
- Check Point #28 - #29

Operator Precedence

- Chapter 2 introduced operator precedence involving arithmetic operators.
  - Example:
    - * and / has higher precedence than + and –
- But what about expressions with other operators
  - Example:
    - `3 + 4 * 4 > 5 * (4 + 3) - 1`
  - What is the value? What is the execution order of the above example?
  - We need to know the precedence rules for this!

Operator Precedence

- Operator precedence and operator associativity determine the order in which Python evaluates operators.
- The expression in the parentheses is evaluated first.
  - Parentheses can be nested, in which case the expression in the inner parentheses is executed first.
- When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule.
- The precedence rule defines precedence for operators.
  - In the next slide, Table 4.7 contains the operators you have learned so far, with the operators listed in decreasing order of precedence from top to bottom.

Operator Precedence

TABLE 4.7 Operator Precedence Chart **(Corrected)**

Precedence	*Operator*
↓	** (Exponentiation)
	+, - (Unary plus and minus)
	*, /, //, % (Multiplication, division, integer division, and remainder)
	+, - (Binary addition and subtraction)
	<, <=, >, >= (Comparison)
	==, != (Equality)
	not
	and
	or
	=, +=, -=, *=, /=, //=, %= (Assignment operators)

Associativity

- If operators with the same precedence are next to each other, their associativity determines the order of evaluation.
- All binary operators except assignment operators are left-associative.
- Example:

  `a - b + c - d` is equivalent to `((a - b) + c) - d`

- Assignment operators are right-associative.
- Example:

  `a = b += c = 5` is equivalent to `a = (b += (c = 5))`

Check Point #28

List the precedence order of the Boolean operators.

- Solution:
  - The decreasing order of precedence order of the Boolean operators:
    1. not
    2. and
    3. or

Check Point #29

Evaluate the following expressions:

1. `True or True and False` → `True`

2. `True and True or False` → `True`

3. `( True and not True ) or False or 10 < 3` → `False`

4. `not ( 20 > 40 and not 40 > 20 ) and True` → `True`

Debugging

Debugging

- Dealing with programming errors:
  - Remember: syntax errors and runtime errors are not difficult to find.
  - However, logic errors can be very challenging.
  - Logic errors are called bugs.
  - Debugging is the process of finding and corrected these logic errors.

Debugging

- Methods of debugging:

1- You can hand-trace the program.

- Meaning, you try to find the error by reading the program
- Clearly, this is very difficult and time consuming.

2- You can insert print statements throughout the program.

- The print statements allow you to see how far the execution has reached.
- You can also print, and then view, the values of variables during execution of the program.
- Again, this is time consuming.

Debugging

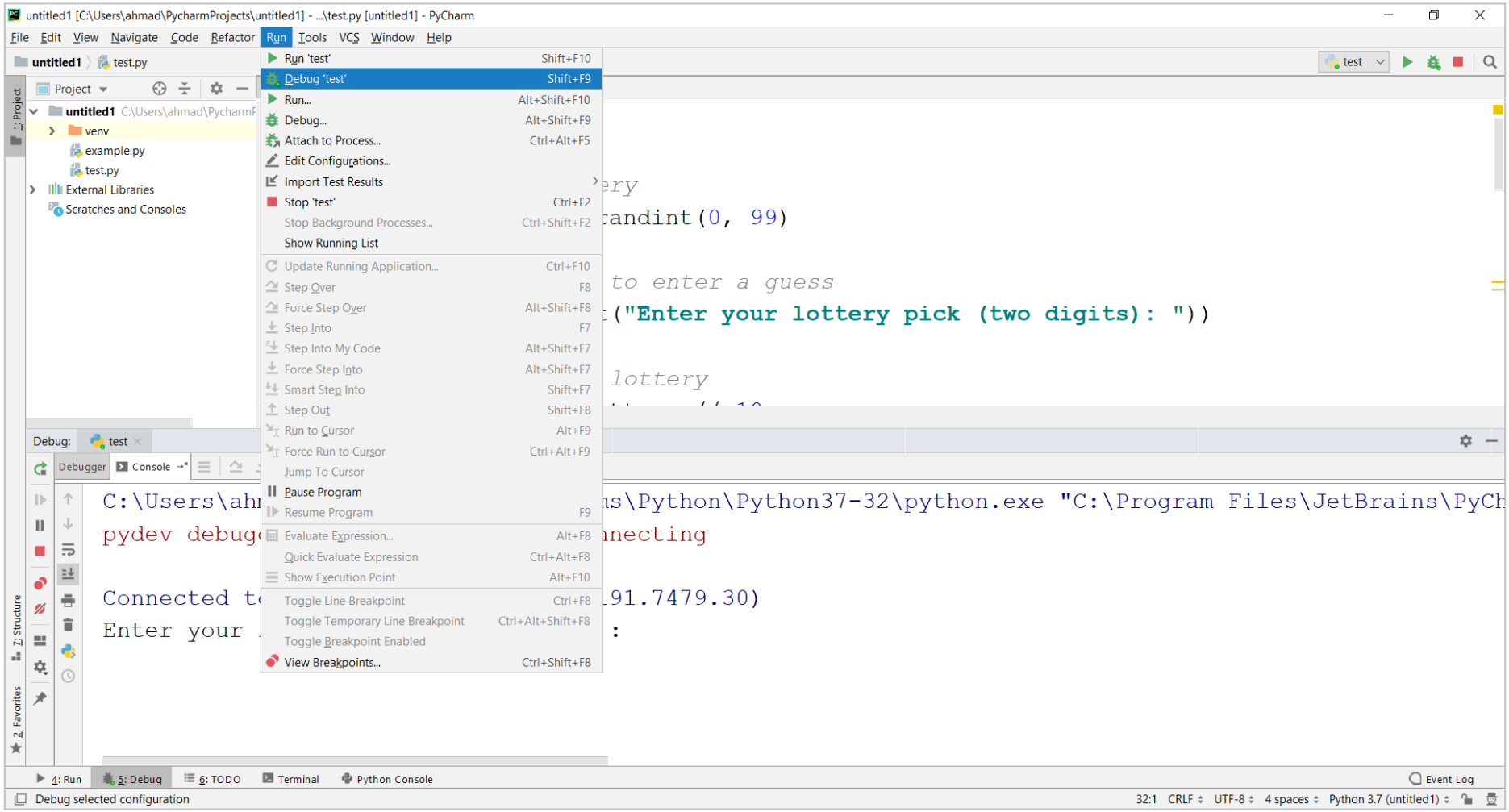
- Methods of debugging:
  - These two methods are okay, but they are slow.
    - They really only work for small, simple programs.
  - So what about large, complex programs?
  - The best solution is to use a debugger utility. (Method 3)

- Most of Python IDE programs, such as PyCharm and IDLE, include integrated debuggers.
  - Learning how to use these debuggers is very important.

Debugging

- Debugger is a program that facilitates debugging.
- You can use a debugger to
  - Execute a single statement at a time.
  - Trace into or stepping over a method.
  - Set breakpoints.
  - Display variables.
  - Display call stack.
  - Modify variables.

Debugging
By Using PyCharm

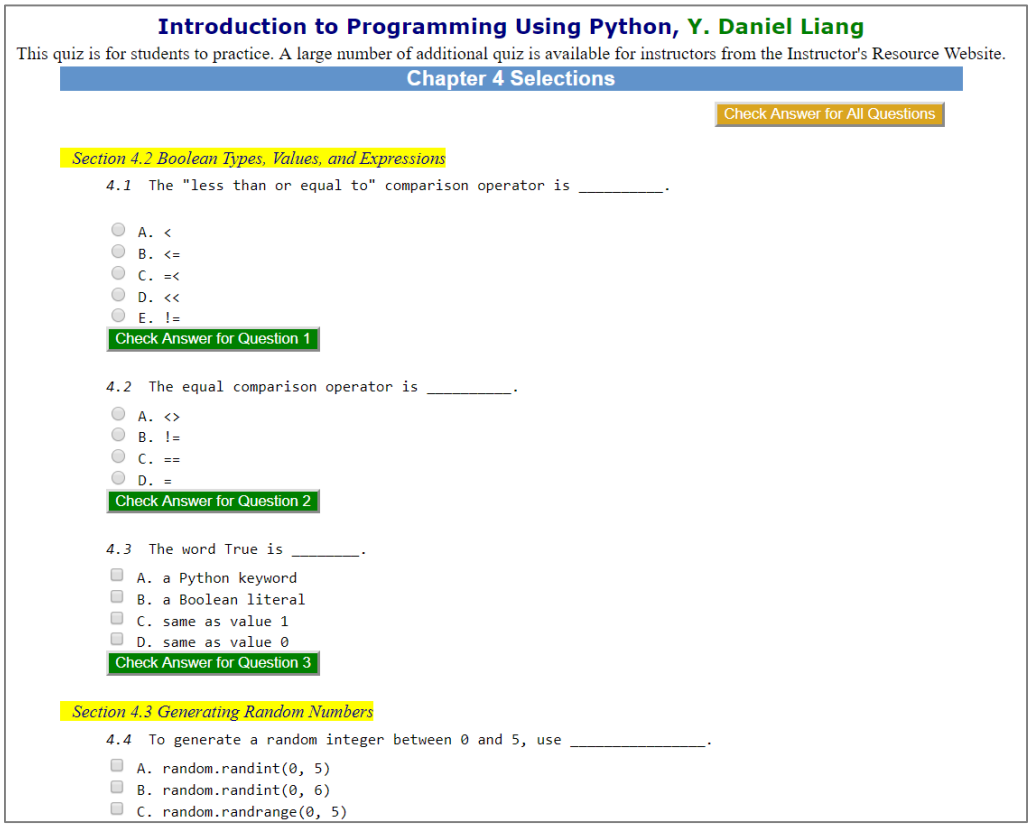


End

- Test Questions
- Programming Exercises

Test Questions

- Do the test questions for this chapter online at https://liveexample-ppe.pearsoncmg.com/selftest/selftestpy?chapter=4



Introduction to Programming Using Python, Y. Daniel Liang

This quiz is for students to practice. A large number of additional quiz is available for instructors from the Instructor's Resource Website.

Chapter 4 Selections

Check Answer for All Questions

Section 4.2 Boolean Types, Values, and Expressions

4.1 The "less than or equal to" comparison operator is _____.

- A. <
- B. <=
- C. =<
- D. <<
- E. !=

Check Answer for Question 1

4.2 The equal comparison operator is _____.

- A. <>
- B. !=
- C. ==
- D. =

Check Answer for Question 2

4.3 The word True is _____.

- A. a Python keyword
- B. a Boolean literal
- C. same as value 1
- D. same as value 0

Check Answer for Question 3

Section 4.3 Generating Random Numbers

4.4 To generate a random integer between 0 and 5, use _____.

- A. random.randint(0, 5)
- B. random.randint(0, 6)
- C. random.randrange(0, 5)

Programming Exercises

- Page 120 – 132:
  - 4.1 - 4.15
  - 4.17 - 4.21
  - 4.24
  - 4.30
- Lab #6