

## Week 6 Python Activities Decisions

### Contents

Week 6 Python Activities Decisions .....	1
Online Tutorials.....	1
DRY.....	2
Control Structures .....	2
Boolean Expression .....	2
Tutorial 1: The if Statement .....	3
Tutorial 2: The else Statement.....	4
String Comparison.....	6
Tutorial 3: The elif Statement .....	6
Tutorial 4: Guessing Game .....	8
Assignment 1: Club Bouncer for Taylor Swift .....	10
TODO .....	11
Assignment Submission.....	11

Time required: 60 minutes.

### Online Tutorials

Please go through these short online tutorials.

- [Python Booleans](#)
- [Python If ... Else](#)
- [Python Introduction](#)
- [Python Get Started](#)
- [Python Syntax](#)
- [Python Comments](#)
- [Python Variables](#)
  - [Variable Names](#)

- [Assign Multiple Values](#)
  - [Output Variables](#)
  - [Global Variables](#)
- [Python Data Types](#)
- [Python Numbers](#)
- [Python Casting](#)
- [Python Strings](#)
- [Python Operators](#)
- [Python User Input](#)

## DRY

**Don't Repeat Yourself**

## Control Structures

A control structure in Python helps manage the flow of code execution. There are three main types.

1. **Sequential:** Code runs line by line, executing one after the other.
2. **Selection (Conditional):** **if**, **elif**, and **else** statements allow different paths based on conditions.
3. **Iteration (Loops):** **for** and **while** loops help execute code repeatedly until a condition is met.

## Boolean Expression

A Boolean expression is an expression that is either true or false. True and False are so-called "Boolean values" that are predefined in Python. True and False are the only Boolean values, and anything that is not False, is True.

The following expressions use a comparison operator `==`, which compares two operands and produces True if they are equal and False otherwise:

```
5 == 5
True
5 == 6
False
```

True and False are special values that belong to the class bool; they are not strings:

```
type(True)
<class 'bool'>
type(False)
<class 'bool'>
```

The == operator is one of the comparison operators; the others are:

<b>x == y</b>	x is equal to y
<b>x != y</b>	x is not equal to y
<b>x &gt; y</b>	x is greater than y
<b>x &lt; y</b>	x is less than y
<b>x &gt;= y</b>	x is greater than or equal to y
<b>x &lt;= y</b>	x is less than or equal to y
<b>x is y</b>	x is the same as y
<b>x is not y</b>	x is not the same as y

Although these operations are probably familiar to you, the Python symbols are different from the mathematical symbols for the same operations. A common error is to use a single equal sign (=) instead of a double equal sign (==). Remember that = is an assignment operator and == is a comparison operator. There is no such thing as =< or =>.

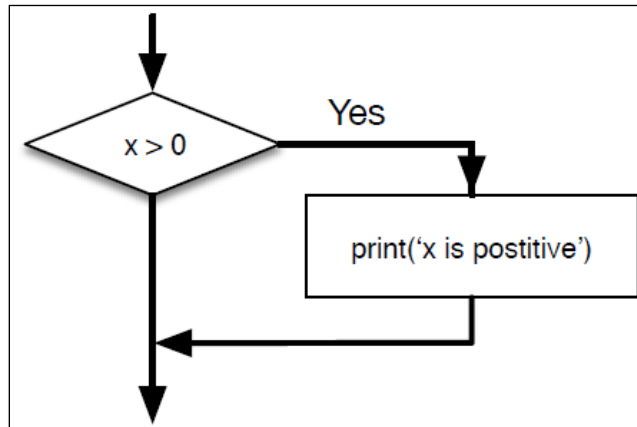
## Tutorial 1: The if Statement

Conditional execution in Python utilizes **if**, **elif** (optional), and **else** statements to execute code based on conditions.

Indentation is crucial; Python uses it to define code blocks.

The expression must be one that will evaluate to either True or False.

1. If the expression evaluates to **True**, the indented statement(s) that follow the colon will be executed in sequence.
2. If the expression evaluates to **False**, those indented statements will be skipped and the next statement following the if statement will be executed.



Create a Python code cell with the following code.

```
1 x = 1
2 # Is x positive or greater than 0?
3 if x > 0:
4     print('x is positive')
```

Example run:

```
x is positive
```

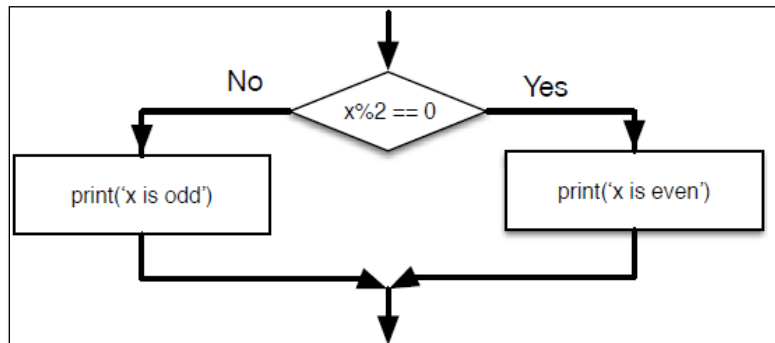
The Boolean expression after the **if** statement is called the condition. We end the **if** statement with a colon character (:) and the line(s) after the if statement are indented.

If the logical condition is **True**, then the indented statement gets executed. If the logical condition is **False**, the indented statement is skipped.

## Tutorial 2: The else Statement

The **else** statement is used in conjunction with **if** statements. It provides a way to specify a block of code that should be executed when the condition of the **if** statement evaluates to **False**.

If the remainder when  $x$  is divided by 2 is 0, then we know that  $x$  is even, and the program displays a message to that effect. If the condition is false, the second set of statements is executed.



Since the condition must either be true or false, exactly one of the alternatives will be executed. The alternatives are called branches because they are branches in the flow of execution.

The **else** statement is optional and helps handle situations when the **if** condition is not met. It's a way to define an alternative action when the condition isn't satisfied.

Create a Python code cell with the following code.

```
1  x = int(input("Enter a number: "))
2
3  # % is the modulo operator
4  # This returns the remainder of integer division
5  # If the remainder is 0, then the number is even
6  if x % 2 == 0:
7      print("x is even")
8  else:
9      print("x is odd")
```

Example runs:

```
Enter a number: 7
x is odd
PS Z:\_WNCC\Python
Enter a number: 8
x is even
```

## String Comparison

Using comparison operators works for strings as well.

```
truck = 'Dodge'
if truck == 'dodge':
    print('You have a Dodge truck.')
else:
    print('You don't have a Dodge truck.')
```

```
You don't have Dodge truck.
```

Python, like many programming languages, is case sensitive. Dodge is not the same as dodge. Use the **.lower()** operator to make sure you are comparing strings properly.

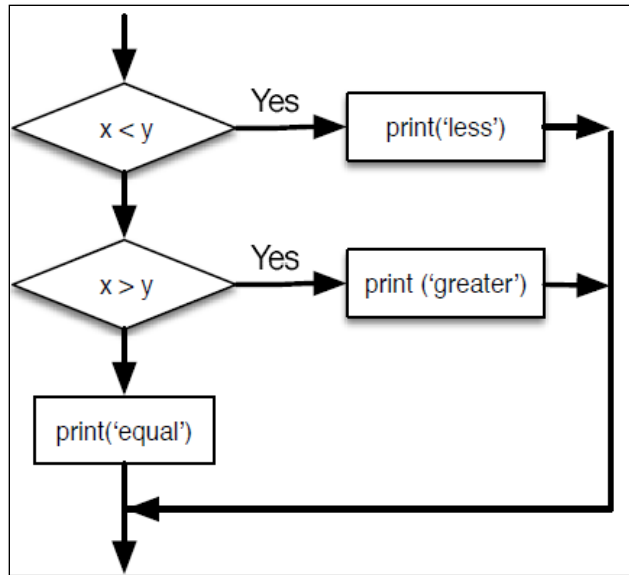
```
truck = 'DoDgE'
if truck.lower() == 'dodge':
    print('You have a Dodge truck.')
else:
    print('You don't have a Dodge truck.')
```

```
You have a Dodge truck.
```

## Tutorial 3: The elif Statement

Here are the three keywords that can be used with **if** statements:

- **if**
- **elif**
- **else**



If there are more than two possibilities, we need more than two branches. One way to express a computation like that is a chained conditional:

```
if x < y:
    print('x is less than y')
elif x > y:
    print('x is greater than y')
else:
    print('x and y are equal')
```

**elif** is an abbreviation of “else if.” Again, exactly one branch will be executed. There is no limit on the number of **elif** statements. If there is an **else** clause, it has to be at the end, but there doesn’t have to be one.

Each condition is checked in order. If the first is false, the next is checked, and so on. If one of them is true, the corresponding branch executes, and the statement ends. Even if more than one condition is true, only the first true branch executes.

This is a use of an **if** statement to assign letter grades. Suppose that scores 90 and above are A’s, scores in the 80s are B’s, 70s are C’s, 60s are D’s, and anything below 60 is an F.

Create a Python code cell with the following code.

```

1  """
2      Name: grades.py
3      Author:
4      Created:
5      Purpose: Determine a letter grade from a score
6  """
7
8  # Get score from the user
9  score = int(input("Enter your score: "))
10
11 # Determine what the grade is and display it
12 if score >= 90:
13     print("A")
14 elif score >= 80:
15     print("B")
16 elif score >= 70:
17     print("C")
18 elif score > 60:
19     print("D")
20 else:
21     print("F")

```

Example run:

```

Enter your score: 87
B

```

Using **elif**, as soon as we find where the score matches, we stop checking conditions and skip all the way to the end of the whole block of statements.

You will rarely use more than one **if** statement in a multiple condition test. Use **elif** as shown above.

## Tutorial 4: Guessing Game

Let's try a guess-a-number program. The computer picks a random number, the player tries to guess, and the program tells them if they are correct. To see if the player's guess is correct, we need an if statement.

The syntax of the **if** statement is a lot like the for statement in that there is a colon at the end of the if condition and the following line or lines are indented. The lines that are indented will be executed only if the condition is true. Once the indentation is done with, the **if** block is concluded.



Create a Python code cell with the following code.

```
1  """
2      Name: guessing_game.py
3      Author:
4      Created:
5      Purpose: Demonstrate if else and random numbers
6  """
7
8  # Import the random library
9  from random import randint
```

To create random numbers, we need the randint function from the random library.

```
11 # Generate a random integer between 1 and 10 inclusive
12 random_num = randint(1, 10)
```

- **randint()** - Generates a random integer between 1 and 10, inclusive. Inclusive means that the range of numbers will include 10.
- **random\_num =** - Assigns the random integer the **random\_num** variable.

```
14 # Prompt and get input from the user
15 guess = int(input("Enter your guess between 1 and 10: "))
```

This line of code in Python allows the user to enter a number as a guess:

- **input("Enter your guess between 1 and 10: ")**: - Displays the message "Enter your guess between 1 and 10: " and waits for the user to input something.
- **int()** - Converts the user's input into an integer (whole number) format.
- **guess =** - Assigns the converted integer value entered by the user to a variable named **guess** which the program can use later.

This line prompts the user to input a number between 1 and 10, converting that input into an integer and storing it in the variable `guess` for further use in the program.

```
17 # Is the guess correct?
18 if guess == random_num:
19     # The guess is correct
20     print("You got it!")
```

This line of code checks if the variable **guess** is equal to the variable **random\_num**. If they match, the expression evaluates as **True**. The code prints "You got it!".

```
21     else:
22         # The guess is incorrect
23         print(f"Sorry, the number is {random_num}")
```

If **guess** is not equal to **random\_num**, the expression evaluates as **False**. Execution goes to the **else** branch. It will print a message saying, "Sorry, the number is" followed by the actual value of **random\_num**, indicating that the guess did not match the randomly chosen number.

```
25     # This last statement is always executed,
26     # after the if statement are complete
27     print("Done")
```

Example program run:

```
Enter your guess between 1 and 10: 5
Sorry, the number is 8
Done
```

## Assignment 1: Club Bouncer for Taylor Swift

Taylor Swift is coming to town. She is playing at the local school auditorium. You want to buy a ticket. Before you can do that, you must write a program to determine who can enter the concert.

When you are testing a single variable for a range of values, when you have eliminated the upper range, you don't have to test that any more.

For example: If someone is not older than 21, you don't have to test again for not being older than 21.

**NOTE:** You can't have more than one age. You would not use multiple if statements. We will rarely if ever use multiple if statements for a single variable.

1. Ask for the user's age.
2. If they are over 21 → they can enter and have a drink.
3. Else if they are between 18 and 21 inclusive (18, 19, 20, or 21) → They can enter but can't have a drink. They must wear a wristband.

4. Otherwise, they are younger than 18 → They are too young.

---

## TODO

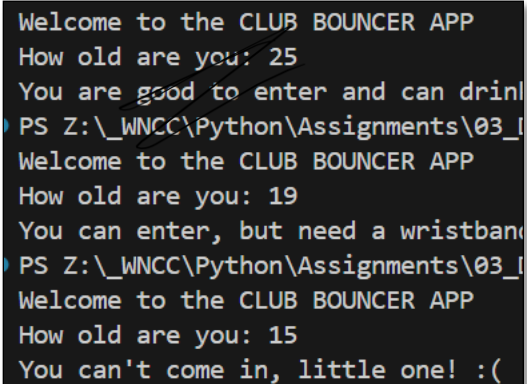
```
# Filename: bouncer.py

# TODO: Print a program title

# TODO: Get user input for age and convert it to an integer

# TODO: Check if the age meets the conditions for entry, print result
# 21+ drink, 18+ can enter, <18 can't enter
```

Example run:



```
Welcome to the CLUB BOUNCER APP
How old are you: 25
You are good to enter and can drink
PS Z:\WNCC\Python\Assignments\03_I
Welcome to the CLUB BOUNCER APP
How old are you: 19
You can enter, but need a wristband
PS Z:\WNCC\Python\Assignments\03_I
Welcome to the CLUB BOUNCER APP
How old are you: 15
You can't come in, little one! :(
```

---

## Assignment Submission

- Insert screenshots of your Google Collab Notebook.
- Insert a Share Link for your Google Collab Notebook. (Test it in another browser to make sure it works without being signed in.)
- Submit in Blackboard.