www.thegordon.edu.au

Python Programming

Section 4 – Selection

UEE60411 Advanced Diploma of Computer Systems Engineering

UEENEED103A Evaluate and modify Object Oriented code programs

For further information:

Steve Gale
ICT – Science and Technology
Ph: (03) 5225 0940
Email: sgale@gordontafe.edu.au

1 – IF statements

So far we have been looking at the first of the three main programming structures: SEQUENCE.

We are now going to look at the second: SELECTION.

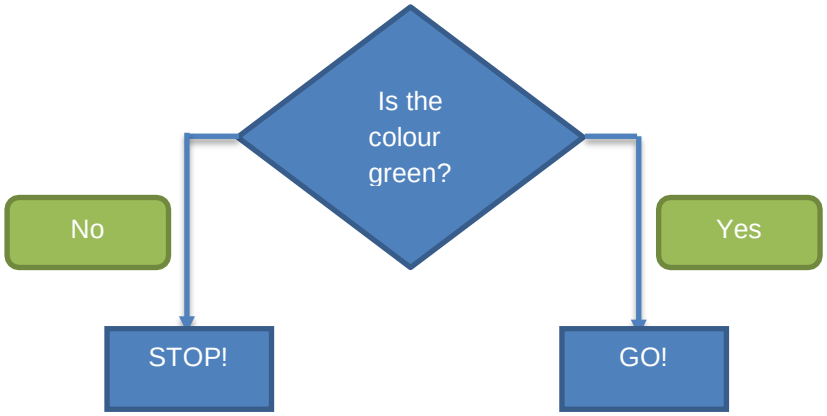
Any but the simplest of computer programs require a method of selecting from two or more options – in other words, a way for the program to make a decision.

This will always be based on the evaluation of a condition. The condition will be a question asked of the program which will give a response of either Yes or No, or True or False.

Below is an example of a simple condition to be evaluated, and the subsequent branches and actions taken depending on the result of that evaluation. First is a flow chart showing how the decision is structured, followed by a real code example that you could use to create a program to display the effect (it is a good idea to actually put the code into a real program).

Note the matching opening and closing brackets for each if and else part of the statement.

Example 1: If colour is green then go, otherwise stop.



(below is the code sample for Example 1)

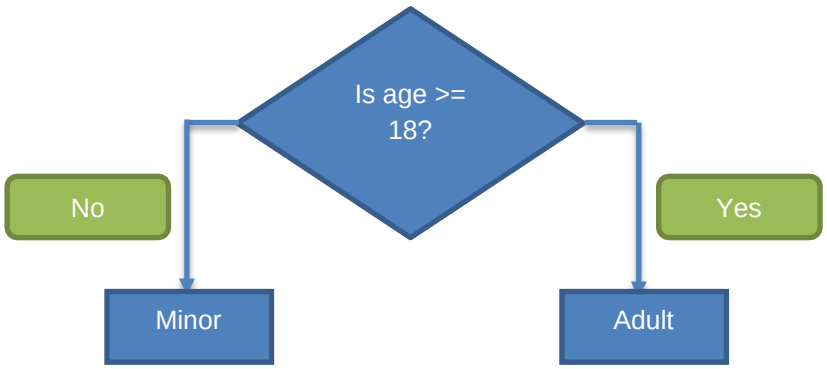
```
1  colour = input("Enter the light colour ('red' or 'green'): ")
2  if colour == "green":
3      print("Go")
4  else:
5      print("Stop")
```

Let's break the code down, line-by-line:

1	The program asks the user to enter either "red" or "green", which is then assigned to the variable `colour`. Note we use `=` (a single `=` character) because this is **assigning** a value.
2	This is the **conditional test**. The program is testing the variable `colour` to see if it contains the character string "green". This test will evaluate to either **true** or **false**. Note we use `==` (double `=` character) because this is **testing** a value. Note also that the line ends with a "`:`".
3	This is the code executed if our line #2 **conditional test** evaluates to **true**; in other words, if the variable `colour` holds the value "green". We print the word "Go" to the output console. Note that this line is **indented** below the `if` statement. Python uses **indentation** to collect code statements together. You'll see later that all the code statements inside another Python structure (in this case, an `if` statement) need to be at the same level of **indentation**.
4	This is the alternative, and only executes if the line #2 **conditional test** evaluates to **false**; in other words, if the variable colour does **not** hold the value "green". Note that this line also ends with a "`:`".
5	Finally, this is the code executed if the line #2 **conditional test** evaluates to **false**. If the variable `colour` does **not** contain the value "green", then **this** line will execute, printing "Stop" to the output console.

Create a new project called "**Handout04**" and then create the above program, calling it **lights01.py**. Enter the code and run the program, ensuring it functions correctly. Try entering different values into the program when it asks for input and see what the results are.

Example 2: If age is equal to or greater than 18, display "Adult", otherwise display "Minor".



(below is the code sample for Example 2)

```
1  age = int(input("Enter age: "))
2  if age >= 18:
3      print("Adult")
4  else:
5      print("Minor")
```

This program functions almost exactly the same as **Example 1**. The only real differences are in lines #1 and #2.

Because we want a numeric value, and not a character string, we need to convert the user's input into a number, which we do using the `int()` function we looked at last class.

Secondly, we're not testing to see if the variable `age` is **equal** to 18, we're testing to see if it is **greater than or equal to**, meaning we use the **comparison operator** `>=`.

Here the program checks to see if the age variable is greater than, or equal to, the value 18. If it is, it will execute line #3 and print "Adult" to the console. If not, it will execute line #5 and print "Minor" to the console.

HINT: Many tutorials, textbooks, and other publications refer to two values being compared, assigned, or mathematically manipulated as **operands**. In the code `c = a + b`, for example, all three variables could be referred to as **operands**.

Comparison operators are used all the time in programming. They test the relationship between values (and are sometimes called **relational operators**). Because they are just testing values, and not calculating anything, they always evaluate to either **true** or **false**. Below is a list of the most commonly-used ones in Python:

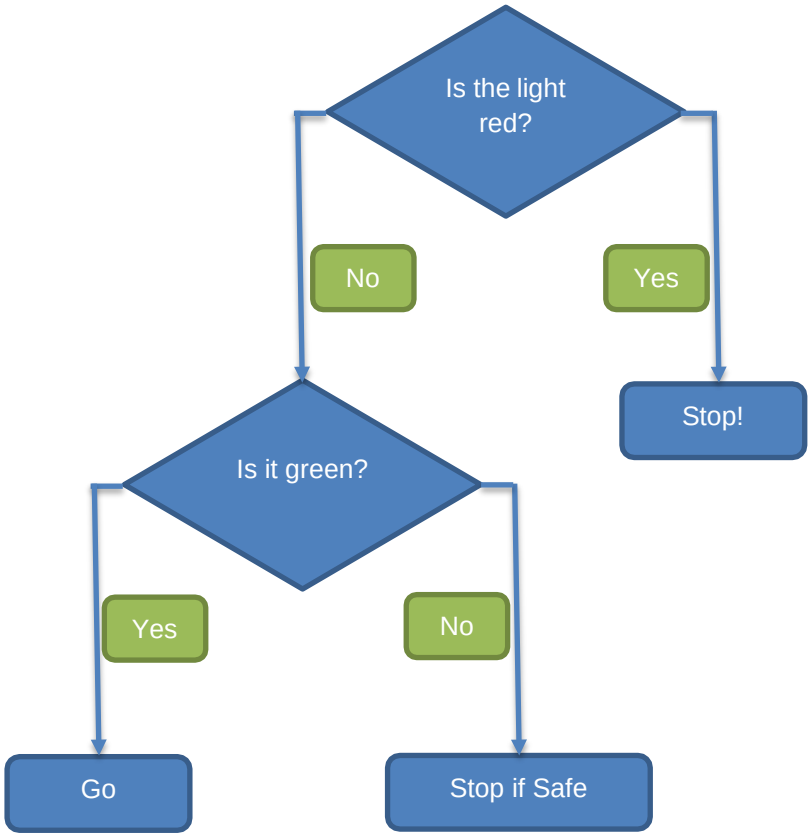
Operator	Explanation	Example
`==`	If the values of two variables are equal, the condition is **true**	`(a == b)` **False**
`!=`	If the values of two variables are not equal, the condition is **true**	`(a != b)` **True**
`<>`	If the values of two variables are not equal, the condition is **true**	`(a <> b)` **True** This is the same as `!=`
`>`	If the value of the left variable is greater than the value of the right variable, the condition is **true**	`(5 > 10)` **False**
`<`	If the value of the left variable is less than the value of the right variable, the condition is **true**	`(5 < 10)` **True**
`>=`	If the value of the left variable is greater than or equal to the value of the right variable, the condition is **true**	`(5 >= 10)` **False**
`<=`	If the value of the left variable is less than or equal to the value of the right variable, the condition is **true**	`(5 <= 10)` **True**

Create the above program, calling it **age01.py**. Enter the code and run the program, ensuring it functions correctly. Try entering different values into the program when it asks for input and see what the results are.

ELIF statements

Sometimes referred to as a "multi-way IF statement", or an "ELSE…IF statement", an `elif` is used when we want to test multiple possible conditions, and provide different outcomes depending on the results.

To look again at our traffic lights example, let's imagine a slightly more complicated option. We want to test to see if the light is red, or green, **or orange**, and we'll need different results for each.



The code for this will look like the following.

```
1  colour = input("Enter the light colour ('red','green', or 'orange'): ")
2  if colour == "red":
3      print("Stop")
4  elif colour == "green":
5      print("Go")
6  else:
7      print("Stop if safe")
```

Create this as **lights02.py** and run it, experimenting with different user inputs. What happens if you put "flashing" in for the light colour? Or "purple"? Can you think of a way to stop a poor output for this?

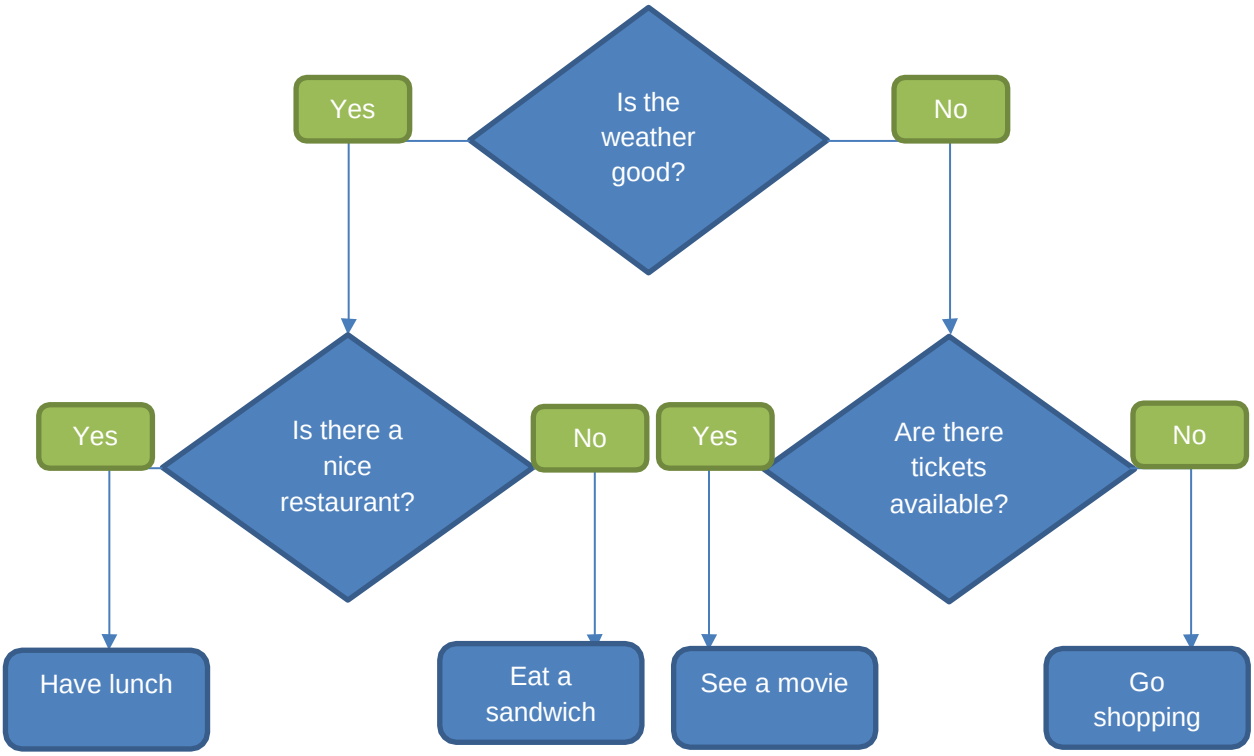
Nested IF statements

A nested IF is used when there is more than two possible outcomes for our test condition. Effectively, what we do is test the condition, then test any *other* conditions that may arise.

Let's say we want to decide what to do today, depending on the weather, the presence of a nice restaurant, and tickets available for the movies.

If the weather is nice, *and* we can find a nice restaurant, we will have lunch. If the weather is nice but we can't find a restaurant, we'll eat a sandwich. If the weather *isn't* nice, and we can get movie tickets, we'll go to the cinema. If the weather isn't nice, and we can't get movie tickets, we'll go shopping.

To accomplish this, we need to only respond to some conditions **if the preceeding conditions occur**. To look at a flowchart of this:



Let's look at the code for this, which uses **nested if** statements.

```
1   weather = input("Is the weather good? ('y' or 'n'): ")
2   if weather == "y":
3       restaurant = input("Is there a nice restaurant ('y' or 'n'): ")
4       if restaurant == "y":
5           print("Have lunch")
6       else:
7           print("Eat a sandwich")
8   else:
9       tickets = input("Are there tickets available? ('y' or 'n'): ")
10      if tickets == "y":
11          print("See a movie")
12      else:
13          print("Go shopping")
14
```

You can see how there are two **nested if** statements (lines #4 and #10). These only execute if their parent **if** statement evaluates to **true**. The if statement on line #4, for example, **only** executes if the weather variable is equal to "y". If it isn't, then the program bypasses lines #3 – 7 entirely, passing execution down to line #8 where it asks about the tickets.

If the user has answered "n" to the weather being good, then we don't need to execute any of that code – it becomes irrelevant, so it's a waste of resources to execute it. We're only interested in the food question if the weather is good.

Sometimes we need to test multiple conditions, one after the other, but we want the program to stop executing once it has found a **true** evaluation.

The program below, **age02.py**, demonstrates this. We want to display the age category of the user depending on their entered age. However, we don't want to run unnecessary tests once we have found the right category. By using consecutive `elif` statements, we can eliminate categories "from the top down". So if the user enters 55 as their age, for example, we can test to see if their age is over 65, which it isn't. So the program will drop to the first `elif` and test to see if it's over 35, which is is. So the code within that `elif` will execute, and the rest of the statement will not execute.

If the entered age was younger, the statements would continue to execute until they found a positive result, at which time the program would finish.

Try creating the program and seeing if you get the results you expect.

```
age = int(input("Enter age: "))
if age > 65:
    print("Elderly")
elif age > 35:
    print("Middle aged")
elif age > 17:
    print("Adult")
elif age > 12:
    print("Teenager")
else:
    print("Child")
```

Simple IF exercises

1. Create a Python program called **branch01.py**
   Input a 4-digit number representing a time in 24-hour format (eg: 1330 = 1:30 pm).
   Use an if statement to display a message indicating if the time is “Morning” or “Afternoon”

```
Enter time in 24 hour format (eg: 1330): 1100
Morning
```

2. Create a Python program called **branch02.py**
   Input two integers. Display a message indicating which of the two numbers is the larger, and which is the smaller.

```
Enter first number: 5
Enter second number: 10
10 is greater than 5
```

3. Create a Python program called **branch03.py**
   Enter the name of a season (eg: summer, autumn, spring, or winter)
   Use **four separate** `if` statements to display a message indicating which months are in the season entered (summer = December, January, February; autumn = March, April, May; winter = June, July, August; spring = September, October, November)
   (**HINT:** you can ensure the case of the entry doesn’t matter by using the `string_name.upper()` method, which will convert the entire string into upper case. Eg: if your variable is `season`, using `season.upper()` will convert “winter” or “Winter” into “WINTER”, no matter what case the user enters).

```
Enter the season: winter
June, July, August
```

4. Create a Python program called **branch04.py**
   The user should enter the name of a day of the week (eg: “Sunday” or “Monday”).
   Use seven (7) `if` statements to indicate if this is a **Weekday** or a **Weekend**.

```
Enter the day: Tuesday
Weekday
```

5. Create a Python program called **branch05.py**
   The user should input the number of days in a month into an integer variable called `days` (remember to use `int()` to convert the keyboard input into a number).
   Use if statements to construct a message giving the names of those months which have that number of days. Display this message in the console.

```
Enter the number of days: 31
January, March, May, July, August, October, December
```

The following exercises will require either `elif` or nested `if` statements.

6. Create a Python program called **branch06.py**
   Input the bank balance for an investment account.
   Use an `if` statement to set the correct interest rate by assigning it to a variable.
   If the balance is less than $1000.00, interest is 5%
   If the balance is >= $1000.00, and < $10,000.00, interest is 5.5%
   If the balance is >= $10,000.00, interest is 6%
   Calculate the interest earned in a year, and display the starting balance, interest rate used, amount of interest earned, and the final balance (starting balance + interest earned).
   Display the resutl on the console.
   (**HINT:** Remember the formatting option to display numbers with the thousands separator.
   Displaying the final balance would be something like:
   `print("Final balance: ${0:,.2f}".format(final_bal))`

```
Enter starting balance: 1500

Starting balance: $1,500.00
Interest rate: 5.50%
Interest earned: $82.50
Final balance: $1,582.50
```

7. Create a Python program called **branch07.py**
   This should enable the user to enter one of four integers: 1, 2, 3, or 4.
   Use an if statement to display the words ONE, TWO, THREE, or FOUR, depending on what integer was entered.

```
Enter the number 1, 2, 3, or 4: 2
TWO
```

8. Modify **branch07.py** and name it **branch08.py**
Include some simple range checking to ensure that **only** the numbers 1 – 4 can be entered. If an incorrect number is entered, your program should display an error message instead.
(HINT: This can be achieved by including an `else:` clause in your `if` statement).

```
Enter the number 1, 2, 3, or 4: 6
ERROR - You must enter a number between 1 and 4
You entered 6
Please run the program again!
```

9. Create a program called **branch09.py**
This should allow the user to enter an amount representing their gross annual income. Your program should use an if statement (similar to **branch06.py**) to calculate the appropriate tax rate to apply, as follows:

Gross Income	Tax Rate
0 – 7999.99	0%
8000 – 14999.99	25%
15000 – 24999.99	35%
25000 – 49999.99	42%
50000 +	48%

Your program should calculate the tax payable, and the nett income.
It should then display the Gross Income, Tax Rate, Tax Payable, and Nett income as below.

```
Enter your gross income: 27500

Gross income: $27,500.00
Tax rate: 42.00%
Tax payable: $1,155.00
Nett income: $26,345.00
```