**Chapters 1-6**

**Functions:**

* **Logical operators: And**; **or**; **not**; (returns True if the argument is false and false when it is true)
* **type**() : will tell you the type of function in the brackets
* **Range**( start, stop, [step]): produces sequence of integers
* **Separator**: sep = ‘\_\_’ : put at the end of a string; for separators between words e.g. “-“ will be put between every word
* **Int()** ; int function takes a certain value and converts it to integer
* **Float**: converts integers and strings (when possible) into floating point numbers.
* **Str():** same as float but for string
* **Input()**: variable = input(prompt) e.g. name = input(‘What is your name?’) what is your name <— then type your name and variable assigned to name.
* **If**: e.g. : x = int(input(‘Enter the value of x: ‘))

if x > 0

print (‘X is positive’)

* **if**-**else**: it is the same as if but with with a secondary condition, that is an else statement.
* **If-elif-else:** it is a set if else-if, that is, allows multiple ifs.
* **In**: use when you want to include a list of items.
* **Random.randint()**: returns a random integer value between 2 higher and lower limits (including both limits) provided as 2 parameters: randint(beg,end). **it is necessary to use import random to use this function.**
* **Help():**gives us information about a function, data type or a module.
* **Dir():**returns list of the attributes and methods of any object (say functions , modules, strings, lists, dictionaries etc.)

**Format():** e.g.format(99.87, ‘.1f’) 🡪 99.9

**Boolean expressions**: (return true or false statements)

* == : Equal to: e.g. 4 == 4 —> **true**; 4 == 73—> **false**
* != : different from
* > greater than
* < smaller than
* >= greater than or equal to
* <= less than or equal to

**Ch.3(previous stuff on this page is not necessarily after ch3)**

**Mathematical operators:**

* \* : multiplication
* // : floor division (divides two numbers and rounds down to an integer: >>> 56/3 = 18.6666666
* \*\* : exponential : elevates a number to a power
* % modulus: divides two integers and returns the remainder of the division e.g. 16%3 = 1. to create or use multiples in python, use the **modulus %,** making it equal to 0 e.g. for a multiple of 7, if num%7 == 0:

Chapter 5: decision making structures and Boolean expressions:

**Loops (general):**

* Loops controlled by a condition use a true/false condition to control the number of repetitions, while a loop controlled by a counter is repeated a given number of times.
* **pass** **statement** in loops: to avoid syntax errors when running an incomplete program. Simply does nothing

**The while loop:**

* **While** statement is used to write a loop controlled by a condition. In a while loop, the repetition of the code is executed until the while condition is true.
* **As long as the condition is true, the block of statements under the while condition is re-executed.** When the condition becomes false, the program execution continues outside the loop.
* **Example:**

Countdown = 5

While countown >= 1:

**Print(countdown)**

**Countdown = countdown – 1**

Print(“go”)

Output : 54321 Go

* The **bold** are the while statements.
* The countdown variable is defined and initialized before the while loop (i.e. the control of the while condition starts with the countdown variable equal to 5).
* The countdown variable is then decremented inside the loop **at each iteration (every time the condition is true). As long as this condition is true, the sequence of statements is indeed reexecuted and the last statement decrements the countdown variable by one unit.**

**The infinite while loop:**

**(The while-if loop)**

* A while loop can be executed infinite times, until a specific condition occurs. The while True syntax can be used instead of the simple while when the loop doesn’t have an easy end to define. **Thus, we don’t have to initialize a variable before entering the loop, but one needs to go add an if containing the exit condition for the loop, and break is the statement that is executed when the if condition occurs. The break statement is used to force the exit from the loop at a given time.**
* P.**88 for example**

**The for loop:**

* **For** loop is used to write a loop controlled by a counter. In a for loop, the repetition of the code is executed a predefined number of times.
* In for loops, a counter variable is used to count the number of loop iterations. The for clause together with the name of the *counter variable ,* **the in keyword**, followed by the colon to close the statement. A sequence of values stands within the square brackets and each value is separated from the other by a comma (a list).
* A statement or a set of statements that is executed at each loop run.
* Hence a for statement is executed as follows: the first value of the list is assigned to the counter variable and the statements within the block are executed. At this point, the next value of the list is assigned to the counter variable and the statements within the block are re-executed again. This process goes on until the last value of the list is assigned to the counter variable.

**Example**: same as before but now with for loop:

For countdown in [5,4,3,2,1]

Print(countdown)

Print(‘Go’)

Output: 54321 Go.

**The range function:**

Range(start, stop, step)

The range function is often used in conjunction with for loops instead of a list to iterate a set of integer values. For instance, the range function range(1,5,2) can be used in a for loop to print the integer values 1,3 as follows:

For I in range (1,5,2):

Print(i)

For a negative step: i.e. the countdown function

For i in range (5,0,-1):

Print(i)

Print(‘stop!’)

The range function can also be adopted inside while loops.

I = 1

While I in range(1,5)

Print(i)

I = I + 1

This program prints the integers 1,2,3,4.

THE RANGE FUNCTION WORKS ONLY WITH INTEGER VALUES, NOT FLOATS.

**Nested loops:**

Example:

For a in range(1,4):

Print(’\*\*\*\*\*’,a)

For b in range (3,0,-1):

Print(‘’,b)

Print(‘stop’)

Output:

\*\*\*\* 1

1

1

1

\*\*\*\* 2

2

2

2

\*\*\*\* 3

3

3

3

Stop

* **Break** statement: **break** lets you terminate the execution of the nearest enclosing loop, and to move the program control outside the loop. i.e. the loop stops when you have done what you want to do.

e.g. for friend in [a,b,c,d,e,f]

if friend == ‘d’

print(“I’ve found him”)

break

print(“Now I’m reading:”, friend)

print(“END”)

Output:

Now I’m reading: a

Now im reading: b

I’ve found him!

End

* **Continue** statement: rejects all the remaining statements in thecurrent iteration and moves the control back to the top of the loop