Download (right-click, save target as ...) this page as a jupyterlab notebook from: Lab5

Laboratory 5: Sequence, Selection, and Repetition - Oh My!

Medrano, Giovanni

R11521018

ENGR 1330 Laboratory 5 - In-Lab

In [1]:
```
# Preamble script block to identify host, user, and kernel
import sys
! hostname
! whoami
print(sys.executable)
print(sys.version)
print(sys.version_info)
```

```
DESKTOP-6HAS1BN
desktop-6has1bn\medra
C:\Users\medra\anaconda3\python.exe
3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
sys.version_info(major=3, minor=8, micro=5, releaselevel='final', serial=0)
```

Sequence

Our first structure is sequential. To belabor the concept we will compute a short list of cubes, by cubing each element in a list and placing it into another list. The example below is dumb, but useful to introduce repetition later in the lab.

First an unusual cell, used to reset a notebook - it will clear the workspace. Here we use it so the notebook will work the same for everyone (at least at first).

In [7]:
```
# reset this notebook
%reset -f
# do a manual kernel restart to get the execution count to restart at one
```

In [8]:
```
AList = [0.0,1.0,2.0,3.0,4.0] # Create a list of floats
```

In [9]:
```
BList = [] #empty list to accept values
```

In [10]:
```
position = -1 # set position pointer to -1
```

In [11]:
```
position = position + 1 # increment position
BList.append(pow(AList[position],3)) # append to BList to build list of cubes
```

In [12]:
```
position = position + 1
BList.append(pow(AList[position],3))
```

In [13]:
```
position = position + 1
BList.append(pow(AList[position],3))
```

In [14]:
```
position = position + 1
BList.append(pow(AList[position],3))
```

In [15]:
```
position = position + 1
BList.append(pow(AList[position],3))
```

In [16]:
```
print(BList)
```

```
[0.0, 1.0, 8.0, 27.0, 64.0]
```

Selection

Our next structure is selection, illustrated by a simple example

A council member will not be allowed to vote on an ordinance if his/her attendence at council meetings is less than 75%. Take the following inputs from the user:

```
1. Number of council meetings held.
2. Number of council meetings attended.
```

Compute the percentage of meetings attended

$$\%_{attended} = \frac{Meetings_{attended}}{Meetings_{total}}*100$$

Use the result to decide whether the council member will be allowed to vote or not.

In [22]:
```
# use our simple I/O methods to obtain council persons name
council_name = str(input('enter council person name'))
```

In [23]:
```
# use our simple I/O methods to obtain meeting count
meetings_total = int(input('How many meetings since last vote?'))
```

In [24]:
```
# use our simple I/O methods to obtain meetings attended
prompt_string = 'How many meetings did ' + council_name + ' attend? '
meetings_attended = int(input(prompt_string))
```

In [25]:
```
# compute percent_attendence
percent_attend = 100.0*(meetings_attended/meetings_total) #the 100.0 forces float
```

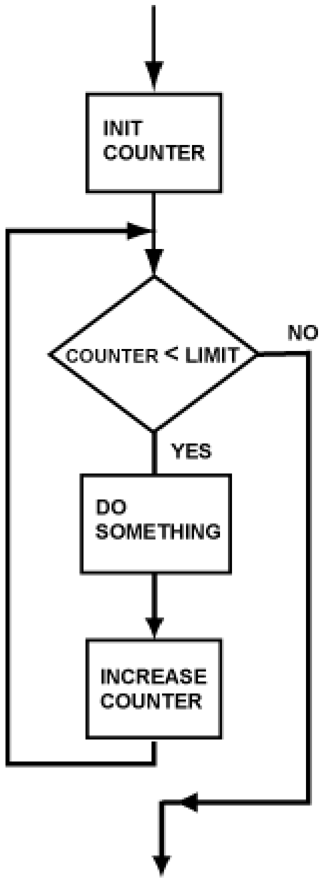
In [26]:
```
# select and make eligibility report
if (percent_attend < 75):
    print('Council person ',council_name,' attended ',percent_attend,' percent of meeti
else:
    print('Council person ',council_name,' attended ',percent_attend,' percent of meeti
```

```
Council person  Rob  attended  70.0  percent of meetings and is NOT eligible to vote
```

Repetition (Loops)

- Controlled repetition
- Structured FOR Loop
- Structured WHILE Loop

Count controlled repetition



Count-controlled repetition is also called definite repetition because the number of repetitions is known before the loop begins executing. When we do not know in advance the number of times we want to execute a statement, we cannot use count-controlled repetition. In such an instance, we would use sentinel-controlled repetition.

A count-controlled repetition will exit after running a certain number of times. The count is kept in a variable called an index or counter. When the index reaches a certain value (the loop bound) the loop will end.

Count-controlled repetition requires

- control variable (or loop counter)
- initial value of the control variable
- increment (or decrement) by which the control variable is modified each iteration through the loop
- condition that tests for the final value of the control variable

We can use both `for` and `while` loops, for count controlled repetition, but the `for` loop in combination with the `range()` function is more common.

Structured `FOR` loop

We have seen the for loop already, but we will formally introduce it here. The `for` loop executes a block of code repeatedly until the condition in the `for` statement is no longer true.

Looping through an iterable

An iterable is anything that can be looped over - typically a list, string, or tuple. The syntax for looping through an iterable is illustrated by an example.

First a generic syntax

```
for a in iterable:
   print(a)
```

Notice the colon `:` and the indentation. Now a specific example:

Example: A Loop to Begin With!

Make a list with "Walter", "Jesse", "Gus, "Hank". Then, write a loop that prints all the elements of your lisk.

In [30]:
```
# # set a list
BB = ["Walter","Jesse","Gus","Hank"]
# # loop thru the list
for AllStrings in BB:
    print(AllStrings)
print('outside da loop')
```

```
Walter
Jesse
Gus
Hank
outside da loop
```

The `range()` function to create an iterable

The `range(begin,end,increment)` function will create an iterable starting at a value of begin, in steps defined by increment (`begin += increment`), ending at `end` .

So a generic syntax becomes

```
for a in range(begin,end,increment):
   print(a)
```

The example that follows is count-controlled repetition (increment skip if greater)

In [31]:
```
# # set a list
```

```
BB = ["Walter","Jesse","Gus","Hank"]
# # Loop thru the list
for i in range(0,4,1): # Change the numbers, what happens?
    print(BB[i])
```

```
Walter
Jesse
Gus
Hank
```

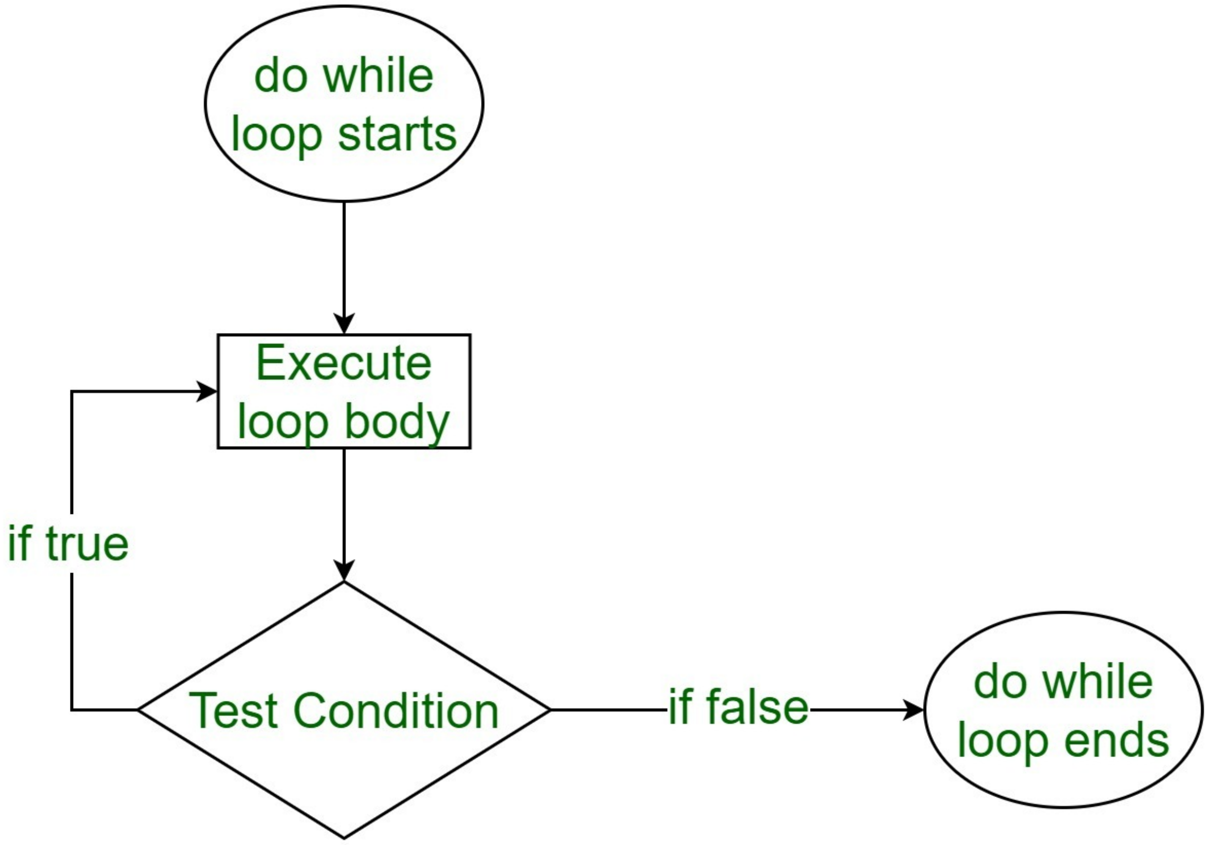
Example: That's odd!

Write a loop to print all the odd numbers between 0 and 10.

In [32]:
```
# # For loop with range
for x in range(1,10,2): # a sequence from 2 to 5 with steps of 1
    print(x)
```

```
1
3
5
7
9
```

Sentinel-controlled repetition



When loop control is based on the value of what we are processing, sentinel-controlled repetition is used. Sentinel-controlled repetition is also called indefinite repetition because it is not known in advance how many times the loop will be executed.

It is a repetition procedure for solving a problem by using a sentinel value (also called a signal value, a dummy value or a flag value) to indicate "end of process". The sentinel value itself need not be a part of the processed data.

One common example of using sentinel-controlled repetition is when we are processing data from a file and we do not know in advance when we would reach the end of the file.

We can use both `for` and `while` loops, for **Sentinel** controlled repetition, but the `while` loop is more common.

Structured `WHILE` loop

The `while` loop repeats a block of instructions inside the loop while a condition remainsvtrue.

First a generic syntax

```
while condition is true:
    execute a
    execute b
....
```

Notice our friend, the colon `:` and the indentation again.

In [33]:
```python
# # set a counter
counter = 5
# # while loop
while counter > 0:
    print("Counter = ",counter)
    counter = counter -1
```

```
Counter =  5
Counter =  4
Counter =  3
Counter =  2
Counter =  1
```

> The while loop structure just depicted is a "decrement, skip if equal" in lower level languages. The next structure, also a while loop is an "increment, skip if greater" structure.

In [35]:
```python
# # set a counter
counter = 0
# # while loop
while counter <= 5:  # change this line to: while counter <= 5: what happens?
    print ("Counter = ",counter)
    counter = counter + 1  # change this line to: counter +=1  what happens?
```

```
Counter =  0
Counter =  1
Counter =  2
Counter =  3
Counter =  4
Counter =  5
```

Beware, its easy to create an infinite loop with this structure. The lab instructor will do so, and illustrate how to regain control of your computer when you do.

Nested Repetition | Loops within Loops

> Round like a circle in a spiral, like a wheel within a wheel
> Never ending or beginning on an ever spinning reel
> Like a snowball down a mountain, or a carnival balloon
> Like a carousel that's turning running rings around the moon
> Like a clock whose hands are sweeping past the minutes of its face
> And the world is like an apple whirling silently in space
> Like the circles that you find in the windmills of your mind!
>
> ***Windmills of Your Mind lyrics © Sony/ATV Music Publishing LLC, BMG Rights Management***
> ***Songwriters: Marilyn Bergman / Michel Legrand / Alan Bergman***
> ***Recommended versions: Neil Diamond | Dusty Springfield | Farhad Mehrad***

"Like the circles that you find in the windmills of your mind", Nested repetition is when a control structure is placed inside of the body or main part of another control structure.

`break` to exit out of a loop

Sometimes you may want to exit the loop when a certain condition different from the counting condition is met. Perhaps you are looping through a list and want to exit when you find the first element in the list that matches some criterion. The break keyword is useful for such an operation. For example run the following program:

In [36]:
```
# #
j = 0
for i in range(0,5,1):
    j += 2
    print ("i = ",i,"j = ",j)
    if j == 6:
        break
```

```
i =  0 j =  2
i =  1 j =  4
i =  2 j =  6
```

In [37]:
```
# # One Small Change
j = 0
for i in range(0,5,1):
    j += 2
    print( "i = ",i,"j = ",j)
    if j == 7:
        break
```

```
i =  0 j =  2
i =  1 j =  4
i =  2 j =  6
i =  3 j =  8
i =  4 j =  10
```

In the first case, the for loop only executes 3 times before the condition j == 6 is TRUE and the loop is exited. In the second case, j == 7 never happens so the loop completes all its anticipated traverses.

In both cases an `if` statement was used within a for loop. Such "mixed" control structures are quite common (and pretty necessary). A `while` loop contained within a `for` loop, with several `if` statements would be very common and such a structure is called **nested control.** There is typically an upper limit to nesting but the limit is pretty large - easily in the hundreds. It depends on the language and the system architecture ; suffice to say it is not a practical limit except possibly for general-domain AI applications.

We can also do mundane activities and leverage loops, arithmetic, and format codes to make useful tables like

Example: Cosines in the loop!

Write a loop to print a table of the cosines of numbers between 0 and 0.01 with steps of 0.001.

In [40]:
```python
import math # package that contains cosine
print("     Cosines     ")
print("   x    ","|"," cos(x) ")
print("--------|--------")
for i in range(0,10,1):
    x = float(i)*0.001
    print("%.3f" % x, "  |", " %.4f "  % math.cos(x)) # note the format code and the pl
```

```
     Cosines
   x    |  cos(x)
--------|--------
0.000   |  1.0000
0.001   |  1.0000
0.002   |  1.0000
0.003   |  1.0000
0.004   |  1.0000
0.005   |  1.0000
0.006   |  1.0000
0.007   |  1.0000
0.008   |  1.0000
0.009   |  1.0000
```

Example: Getting the hang of it!

Write a Python script that takes a real input value (a float) for x and returns the y value according to the rules below

\begin{gather} y = x~for~0 <= x < 1 \\ y = x^2~for~1 <= x < 2 \\ y = x + 2~for~2 <= x < 3 \\ \end{gather}

Test the script with x values of 0.0, 1.0, 1.1, and 2.1.

add functionality to **automaticaly** populate the table below:

x	y(x)
0.0	
1.0	
2.0	
3.0	
4.0	
5.0	

In [41]:
```
userInput = input('Enter enter a float') #ask for user's input
x = float(userInput)
print("x:", x)

if x >= 0 and x < 1:
    y = x
    print("y is equal to",y)
elif x >= 1 and x < 2:
     y = x*x
     print("y is equal to",y)
else:
    y = x+2
    print("y is equal to",y)
```

```
x: 2.5
y is equal to 4.5
```

In [42]:
```
# without pretty table

print("---x---","|","---y---")
print("--------|--------")
for x in range(0,6,1):
    if x >= 0 and x < 1:
        y = x
        print("%4.f" % x, "   |", " %4.f " % y)
    elif x >= 1 and x < 2:
        y = x*x
        print("%4.f" % x, "   |", " %4.f " % y)
    else:
        y = x+2
        print("%4.f" % x, "   |", " %4.f " % y)
```

```
---x--- | ---y---
--------|--------
   0    |     0
   1    |     1
   2    |     4
   3    |     5
   4    |     6
   5    |     7
```

In [46]:
```
# # with pretty table

from prettytable import PrettyTable #Required to create tables
```

```
t = PrettyTable(['x', 'y']) #Define an empty table


for x in range(0,6,1):
    if x >= 0 and x < 1:
        y = x
        print("for x equal to", x, ", y is equal to",y)
        t.add_row([x, y]) #will add a row to the table "t"
    elif x >= 1 and x < 2:
        y = x*x
        print("for x equal to", x, ", y is equal to",y)
        t.add_row([x, y])
    else:
        y = x+2
        print("for x equal to", x, ", y is equal to",y)
        t.add_row([x, y])

print(t)
```

```
for x equal to 0 , y is equal to 0
for x equal to 1 , y is equal to 1
for x equal to 2 , y is equal to 4
for x equal to 3 , y is equal to 5
for x equal to 4 , y is equal to 6
for x equal to 5 , y is equal to 7
+---+---+
| x | y |
+---+---+
| 0 | 0 |
| 1 | 1 |
| 2 | 4 |
| 3 | 5 |
| 4 | 6 |
| 5 | 7 |
+---+---+
```

The `continue` statement

The continue instruction skips the block of code after it is executed for that iteration, and continues with the loop traverse It is best illustrated by an example.

In [47]:
```
j = 0
for i in range(0,5,1):
    j += 2
    print ("\n i = ", i , ", j = ", j) #here the \n is a newline command
    if j == 6:
        continue
    else:
        print(" this message will be skipped over if j = 6 ") # still within the loop,

# #When j ==6 the line after the continue keyword is not printed.
# #Other than that one difference the rest of the script runs normally.
```

```
 i =  0 , j =  2
 this message will be skipped over if j = 6

 i =  1 , j =  4
 this message will be skipped over if j = 6
```

```
i =  2 , j =  6

i =  3 , j =  8
this message will be skipped over if j = 6

i =  4 , j =  10
this message will be skipped over if j = 6
```

The `try` , `except` structure

An important control structure (and a pretty cool one for error trapping) is the `try` , `except` statement.

The statement controls how the program proceeds when an error occurs in an instruction. The structure is really useful to trap likely errors (divide by zero, wrong kind of input) yet let the program keep running or at least issue a meaningful message to the user.

The syntax is:

```
try:
do something
except:
do something else if ``do something'' returns an error
```

Here is a really simple, but hugely important example:

In [48]:
```python
#MyErrorTrap.py
x = 12.
y = 12.
while y >= -12.: # sentinel controlled repetition
    try:
        print ("x = ", x, "y = ", y, "x/y = ", x/y)
    except:
        print ("error divide by zero")
    y -= 1
```

```
x =  12.0 y =  12.0 x/y =  1.0
x =  12.0 y =  11.0 x/y =  1.0909090909090908
x =  12.0 y =  10.0 x/y =  1.2
x =  12.0 y =  9.0 x/y =  1.3333333333333333
x =  12.0 y =  8.0 x/y =  1.5
x =  12.0 y =  7.0 x/y =  1.7142857142857142
x =  12.0 y =  6.0 x/y =  2.0
x =  12.0 y =  5.0 x/y =  2.4
x =  12.0 y =  4.0 x/y =  3.0
x =  12.0 y =  3.0 x/y =  4.0
x =  12.0 y =  2.0 x/y =  6.0
x =  12.0 y =  1.0 x/y =  12.0
error divide by zero
x =  12.0 y =  -1.0 x/y =  -12.0
x =  12.0 y =  -2.0 x/y =  -6.0
x =  12.0 y =  -3.0 x/y =  -4.0
x =  12.0 y =  -4.0 x/y =  -3.0
x =  12.0 y =  -5.0 x/y =  -2.4
x =  12.0 y =  -6.0 x/y =  -2.0
x =  12.0 y =  -7.0 x/y =  -1.7142857142857142
```

```
x =  12.0 y =  -8.0 x/y =  -1.5
x =  12.0 y =  -9.0 x/y =  -1.3333333333333333
x =  12.0 y =  -10.0 x/y =  -1.2
x =  12.0 y =  -11.0 x/y =  -1.0909090909090908
x =  12.0 y =  -12.0 x/y =  -1.0
```

So this silly code starts with x fixed at a value of 12, and y starting at 12 and decreasing by 1 until y equals -1. The code returns the ratio of x to y and at one point y is equal to zero and the division would be undefined. By trapping the error the code can issue us a measure and keep running.

Modify the script as shown below,Run, and see what happens

In [50]:
```
#NoErrorTrap.py
x = 12.
y = 12.
while y >= -12.: # sentinel controlled repetition
    try:
        print ("x = ", x, "y = ", y, "x/y = ", x/y)
    except:
        print('error divide by zero')
    y -= 1
```

```
x =  12.0 y =  12.0 x/y =  1.0
x =  12.0 y =  11.0 x/y =  1.0909090909090908
x =  12.0 y =  10.0 x/y =  1.2
x =  12.0 y =  9.0 x/y =  1.3333333333333333
x =  12.0 y =  8.0 x/y =  1.5
x =  12.0 y =  7.0 x/y =  1.7142857142857142
x =  12.0 y =  6.0 x/y =  2.0
x =  12.0 y =  5.0 x/y =  2.4
x =  12.0 y =  4.0 x/y =  3.0
x =  12.0 y =  3.0 x/y =  4.0
x =  12.0 y =  2.0 x/y =  6.0
x =  12.0 y =  1.0 x/y =  12.0
error divide by zero
x =  12.0 y =  -1.0 x/y =  -12.0
x =  12.0 y =  -2.0 x/y =  -6.0
x =  12.0 y =  -3.0 x/y =  -4.0
x =  12.0 y =  -4.0 x/y =  -3.0
x =  12.0 y =  -5.0 x/y =  -2.4
x =  12.0 y =  -6.0 x/y =  -2.0
x =  12.0 y =  -7.0 x/y =  -1.7142857142857142
x =  12.0 y =  -8.0 x/y =  -1.5
x =  12.0 y =  -9.0 x/y =  -1.3333333333333333
x =  12.0 y =  -10.0 x/y =  -1.2
x =  12.0 y =  -11.0 x/y =  -1.0909090909090908
x =  12.0 y =  -12.0 x/y =  -1.0
```

Readings

Here are some great reads on this topic:

- "Python for Loop" available at https://www.programiz.com/python-programming/for-loop/
- "Python "for" Loops (Definite Iteration)" by John Sturtz available at https://realpython.com/python-for-loop/
- "Python "while" Loops (Indefinite Iteration)" by John Sturtz available at https://realpython.com/python-while-loop/

- "loops in python" available at https://www.geeksforgeeks.org/loops-in-python/
- "Python Exceptions: An Introduction" by Said van de Klundert available at https://realpython.com/python-exceptions/

Here are some great videos on these topics:

- "Python For Loops - Python Tutorial for Absolute Beginners" by Programming with Mosh available at https://www.youtube.com/watch?v=94UHCEmprCY
- "Python Tutorial for Beginners 7: Loops and Iterations - For/While Loops" by Corey Schafer available at https://www.youtube.com/watch?v=6iF8Xb7Z3wQ
- "Python 3 Programming Tutorial - For loop" by sentdex available at https://www.youtube.com/watch?v=xtXexPSfcZg

```
In [ ]:
```