Lab 6: Conditionals and while loop

1 Conditionals

Example 1.

Write a script that determines if a number falls within the interval $[-17, 25]$.

```
x = float(input('What is your number? '))
minVal = -17
maxVal = 25
if x >= minVal and x <= maxVal:
    print( f'Your number {x} is in the interval.')
elif x > maxVal:
    print(f'Your number {x} is bigger than the maximum value {maxVal}.')
else:
    print(f'Your number {x} is smaller than minimum value {minVal}.')

# Run the script to check about your choices.
```

Example 2.

Write a function that takes a number x and returns 10 if $x > -2,\ 1$ if $x < -2$, and 9 if $x = -2$.

```
def my_fun(x):
  if x>2:
    return 10
  elif x<-2:
    return 1
  else:
    return 9

# function usage
my_fun(5) # check with other numbers
```

Note that this is a function example, you can easily convert it to a script.

Example 3.

Write a *script* that asks for a numeric grade from 0 to 100 and returns a letter grade according to the following scheme. Your function should raise an *error* if the grade passed in is not between 0 and 100. Use as few comparisons as possible.

Letter	Criteria
A	$85 \leq$ numeric grade ≤ 100
B	$65 \leq$ numeric grade < 85
F	numeric grade < 65

```
n = float(input('What is your numerical grade? '))

if n < 0 or n > 100:
    raise ValueError(f'Input {n} is not a valid grade.')
   # Raises a ValueError with an error message
elif n < 65:
    grade = 'F'
elif n < 85:
    grade = 'B'
else:
    grade = 'A'

print(f'Your grade is: {grade}')
```

Now run the script and check several numerical grades.

2 while loops

Example 4.

Find the fewest number of terms needed in the series $1 + 2 + 3 + \ldots$ so that the sum > 2010.

```
sum = 0 #initialize sum
n = 0 # initialize n
while sum <= 2010:
    n = n + 1
    sum = sum + n

print(f'The sum of the {n} terms is {sum} which is bigger than 2010.')
```

Run this to get

```
The sum of the 63 terms is 2016 which is bigger than 2010.
```

Example 5.

Write a script that finds the smallest N so that $\ln N > 3$.

```
import numpy as np
N = 1 # initialize N

while np.log(N) <= 3:
 N = N + 1

print(N)
```

You will get 21. (Note $\ln 0$ is undefined, so do not initialize N with 0)

Example 6.

Newton's Method Revisited: Write a script with the given specified initial approximation x_0 to approximate the solution correct up 10 decimal places.

$$x + e^x - 2 = 0, \qquad x_0 = 1$$

```
def my_newt(f, df, x0, Tol):
# Tol stands for tolerance. It represents the acceptable level of
# error in numerical computations.
     x = x0 # initialize x with the intial value
     while abs(f(x)) > Tol:
         x = x - f(x)/df(x)
     print(x)

#function usage

import numpy as np

my_newt(lambda x: x + np.exp(x) - 2, lambda x: 1 + np. exp(x), 1, 1e-10)

0.4428544010040325
```

Example 7.

If we want to see how many iterations the Newton's method was used to find the approximate solution, we can add one more argument in the function as shown below.

```
def my_newton(f, df, x0, Tol, max_iter):
    x = x0
    n = 0  # Initialize the iteration counter

    while n < max_iter:
       if abs(f(x)) < Tol:
          print('Approx. solution found after', n, 'iterations.')
          return x

       if df(x) == 0: # optional
          print('Zero derivative. No solution found.')
          return None

       x = x - f(x) / df(x)
       n += 1  # Increment the iteration counter

    print('Did not converge within', max_iter, 'iterations.')
    return None

# Example usage:
sol = my_newton(lambda x: x+np.exp(x)-2, lambda x: 1+np.exp(x), 1,
 1e-10, 100)
print(f'Approx. solution: {sol}')

Found solution after 4 iterations.
Approx. solution: 0.4428544010040325
```

Example 8.

A slight variation of example 7 is given below.

```
def newton_method(f, df, x0, tol, max_iter=100):
  x = x0
  n = 0
  while abs(f(x)) > tol and n < max_iter:
     x = x - f(x) / df(x)
     n += 1
```

```
  if abs(f(x)) <= tol:
    return x, n
  else:
    return None, n
# Example usage:
approx_zero, num_iterations = newton_method(lambda x: x + np.exp(x)-2,
                                    lambda x: 1 + np.exp(x), 1, 1e-10)

if approx_zero is not None:
print("Approximate zero:", approx_zero)
print("Number of iterations:", num_iterations)
else:
print("Newton's method failed to converge.")
```

Approximate zero: 0.4428544010040325
Number of iterations: 4

---

3 Exercises:

1. Recall that the Fibonacci numbers are defined as follows: (See Lab 4 Exercise #2.)

$$\begin{aligned} F_1 &= 1, \quad F_2 = 1, \\ F_i &= F_{i-1} + F_{i-2}, \quad i \geq 3 \end{aligned}$$

   Write a *Python function* to calculate the nth Fibonacci number, F_n. Your function should produce correct outputs for all n including 1 and 2.

2. Write a function that takes a number x and returns 5 if $x > 1,\ 10$ if $x < 1$, and 100 if $x = 1$.

3. Recall that the compound interest formula:

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

   where P is the principal, r is the interest rate in decimals, t is the time in years, and n is number of times the interest compounds in a year.

   Write a script that asks for the inputs, and calculates the compound interest, A. If the interest rate is not in between 0 and 1, the code should raise an error message.

   (i) $P = \$8,000,\ r = 0.08,\ n = 12, t = 6$
   (ii) $P = \$8,000,\ r = 1.08,\ n = 12, t = 6$

4. Write a *script* that asks for a numeric grade from 0 to 100 and returns a letter grade according to the following scheme. Your function should raise *error* if the grade passed in is not between 0 and 100. Use as few comparisons as possible.

Letter	**Criteria**
A	$90 \leq$ numeric grade ≤ 100
B	$80 \leq$ numeric grade < 90
C	$70 \leq$ numeric grade < 80
D	$60 \leq$ numeric grade < 70
F	numeric grade < 60

5. Create a function for the Newton's method as in **Example 7** to approximate the solution correct up to 10 decimal places using max of 10 iterations.

   (a) $x^2 - 17 = 0, \qquad x_0 = 40$

   (b) $x^2 - 17 = 0, \qquad x_0 = 200$

   (c) $x^5 - 17 = 0, \qquad x_0 = 5$