Hands-on Activity 2.1 : Dynamic Programming

Objective(s):

This activity aims to demonstrate how to use dynamic programming to solve problems.

Intended Learning Outcomes (ILOs):

- Differentiate recursion method from dynamic programming to solve problems.
- Demonstrate how to solve real-world problems using dynamic programming

Resources:

- Jupyter Notebook

Procedures:

1. Create a code that demonstrate how to use recursion method to solve problem

2. Create a program codes that demonstrate how to use dynamic programming to solve the same problem

Question:

Explain the difference of using the recursion from dynamic programming using the given sample codes to solve the same problem

Type your answer here:

3. Create a sample program codes to simulate bottom-up dynamic programming

4. Create a sample program codes that simulate tops-down dynamic programming

Question:

Explain the difference between bottom-up from top-down dynamic programming using the given sample codes

Type your answer here:

0/1 Knapsack Problem

- Analyze three different techniques to solve knapsacks problem

1. Recursion
2. Dynamic Programming
3. Memoization

```
#sample code for knapsack problem using recursion
def rec_knapSack(w, wt, val, n):

  #base case
  #defined as nth item is empty;
  #or the capacity w is 0
  if n == 0 or w == 0:
    return 0

  #if weight of the nth item is more than
  #the capacity w, then this item cannot be included
  #as part of the optimal solution
  if(wt[n-1] > w):
    return rec_knapSack(w, wt, val, n-1)

  #return the maximum of the two cases:
  # (1) include the nth item
  # (2) don't include the nth item
  else:
    return max(
        val[n-1] + rec_knapSack(
            w-wt[n-1], wt, val, n-1),
            rec_knapSack(w, wt, val, n-1)
    )


#To test:
val = [60, 100, 120] #values for the items
wt = [10, 20, 30] #weight of the items
w = 50 #knapsack weight capacity
n = len(val) #number of items

rec_knapSack(w, wt, val, n)
```

```
220
```

```
#Dynamic Programming for the Knapsack Problem
def DP_knapSack(w, wt, val, n):
  #create the table
  table = [[0 for x in range(w+1)] for x in range (n+1)]

  #populate the table in a bottom-up approach
  for i in range(n+1):
    for w in range(w+1):
      if i == 0 or w == 0:
        table[i][w] = 0
      elif wt[i-1] <= w:
        table[i][w] = max(val[i-1] + table[i-1][w-wt[i-1]],
                          table[i-1][w])
  return table[n][w]


#To test:
val = [60, 100, 120]
wt = [10, 20, 30]
w = 50
n = len(val)

DP_knapSack(w, wt, val, n)
```

```
220
```

```
#Sample for top-down DP approach (memoization)
#initialize the list of items
val = [60, 100, 120]
wt = [10, 20, 30]
w = 50
n = len(val)

#initialize the container for the values that have to be stored
#values are initialized to -1
calc =[[-1 for i in range(w+1)] for j in range(n+1)]


def mem_knapSack(wt, val, w, n):
  #base conditions
  if n == 0 or w == 0:
    return 0
  if calc[n][w] != -1:
    return calc[n][w]

  #compute for the other cases
  if wt[n-1] <= w:
    calc[n][w] = max(val[n-1] + mem_knapSack(wt, val, w-wt[n-1], n-1),
                     mem_knapSack(wt, val, w, n-1))
    return calc[n][w]
  elif wt[n-1] > w:
    calc[n][w] = mem_knapSack(wt, val, w, n-1)
    return calc[n][w]

mem_knapSack(wt, val, w, n)
```

```
220
```

Code Analysis

My overall analysis base on the given code that been provided here are the reason that it did use some variables like memo, conditions in order to repeat itself the values until it found the certain value until the argument reach its base case.

Seatwork 2.1

Task 1: Modify the three techniques to include additional criterion in the knapsack problems

```
def rec_knapSack(w, wt, val, d, n):
  if n == 0 or w == 0 or d == 0:
    return 0
  if(wt[n-1] > w):
    return rec_knapSack(w, wt, val, d, n-1,)
  else:
    return max(
        val[n-1] + rec_knapSack(
            w-wt[n-1], wt, val, d, n-1,),
            rec_knapSack(w, wt, val, d, n-1,)
    )
val = [60, 100, 120]
wt = [10, 20, 30]
w = 50
n = len(val)
d = []

for i in range(len(wt)):
  d.append(val[i]/wt[i])

print(rec_knapSack(w, wt, val, d, n))
```

```
220
```

Task 2: Create a sample program that find the nth number of Fibonacci Series using Dynamic Programming

```
  def fiboNum(n):
    if n <=  0:
      return 0
    elif n == 1:
      return 1

    fiboNum_array = [0] * (n + 1)
    fiboNum_array[0] = 0
    fiboNum_array[1] = 1

    for i in range(2, n + 1):
      fiboNum_array[i] = fiboNum_array[i - 1] + fiboNum_array[i - 2]
    return fiboNum_array[n]

  n = 21
  print(f"Fibonacci Sequence value is {n} is also equal to {fiboNum(21)}")
```

Supplementary Problem (HOA 2.1 Submission):

- Choose a real-life problem
- Use recursion and dynamic programming to solve the problem

```
def artmats_knapSack(mw, iw, val, d, n):
  if n == 0 or mw == 0 or d == 0:
    return 0
  if(iw[n-1] > mw):
    return artmats_knapSack(mw, iw, val, d, n-1,)
  else:
    return max(
        val[n-1] + artmats_knapSack(
            mw-iw[n-1], iw, val, d, n-1,),
            artmats_knapSack(mw, iw, val, d, n-1,)
    )
val = [150, 65, 300]
iw = [25, 30, 50]
mw = 80
n = len(val)
d = []
for i in range(len(iw)):
  d.append(val[i]/iw[i])
total_artmats = (artmats_knapSack(mw, iw, val, d, n))
print(f" This is will be the total value of the arts materials in knapsack is: {total_artmats}" )
```

```
 This is will be the total value of the arts materials in knapsack is: 450
```

```
def gymEqmt_knapSack(mw, wt, val, n):

  table = [[0 for x in range(mw+1)] for x in range (n+1)]


  for i in range(n+1):
    for mw in range(mw+1):
      if i == 0 or mw == 0:
        table[i][mw] = 0
      elif wt[i-1] <= mw:
        table[i][mw] = max(val[i-1] + table[i-1][mw-wt[i-1]],
                           table[i-1][mw])
  return table[n][w]
val = [300, 150, 110]
wt = [30, 20, 15]
mw = 180
n = len(val)
total_eqmts = gymEqmt_knapSack(mw, wt, val, n)
print(f" This is be the total value of the gym equipments: {total_eqmts}")
```

```
 This is be the total value of the gym equipments: 560
```

Conclusion

For the overall conclusion i would like to conclude for this hands on activity that I learn how recursion works especially applying it into