Code First Girls Coding Challenge

By Churreesha Harden

Core:Should We Round Up The Final Grade?

Challenge: Every subject is graded from 0 to 100%. Less than 40% is failing grade and more than 80% is a distinction.

We can round up a grade:

If the difference between the grade and the next multiple of 5 is less than 3, round up to the next multiple of 5.

If the value of is less than 40, no rounding occurs as the result will still be a failing grade.

Given a input grade, round it up if appropriate and tell us if the student passed, failed or received a distinction. Write a algorithm and produce a flow chart.

Challenge Problem #1 (Core)

Examples:

round 93% to 95%, report distinction (95 - 93 < 3)

do not round 38% , report fail (38<40)

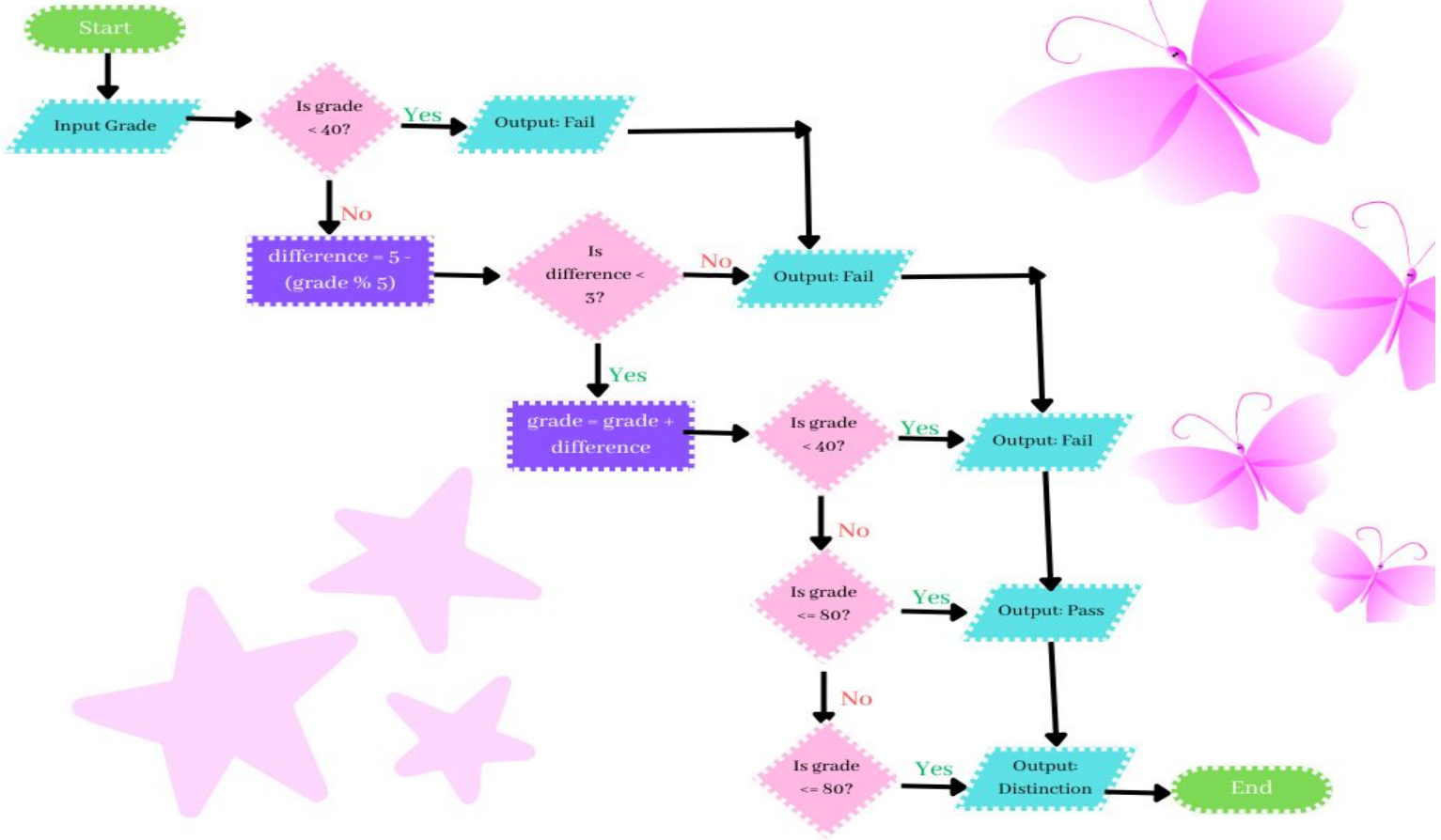
do not round 47%, report pass (50 - 47 = 3)

Should We Round Up The Final Grade? Pseudocode

```
- Define a function to round up a grade and report the result
function round_grade(grade):
-  Calculate the difference between the grade and the next multiple of 5
difference = 5 -  (grade % 5)
- If the difference is less than 3 and the grade is not less than 40, round up the grade
if difference < 3 and grade >= 40:
grade = grade + difference
-  If the grade is less than 40, report fail
if grade < 40:
report = "fail"
- If the grade is between 40 and 80, report pass
else if grade <= 80:
report = "pass"
- If the grade is more than 80, report distinction
else:
report = "distinction"
- Return the rounded grade and the report
return grade, report

-Test the function with some examples
print(round_grade(93)) #  (95, 'distinction')
print(round_grade(38)) #  (38, 'fail')
```

Flow Chart


Start
Input Grade
Is grade < 40?
Yes
Output: Fail
No
difference = 5 - (grade % 5)
Is difference < 3?
No
Output: Fail
Yes
grade = grade + difference
Is grade < 40?
Yes
Output: Fail
No
Is grade <= 80?
Yes
Output: Pass
No
Is grade <= 80?
Yes
Output: Distinction
End

Core: Sorting an array. 🌷

Challenge:You have an array of maximum size of 100 with DISTINCT integers. Write a algorithm and produce a flow chart that sorts this array from smallest to largest.

EXAMPLE:

[1, 4, 5, 66, 3, 84, 11, 198]

SORTED:

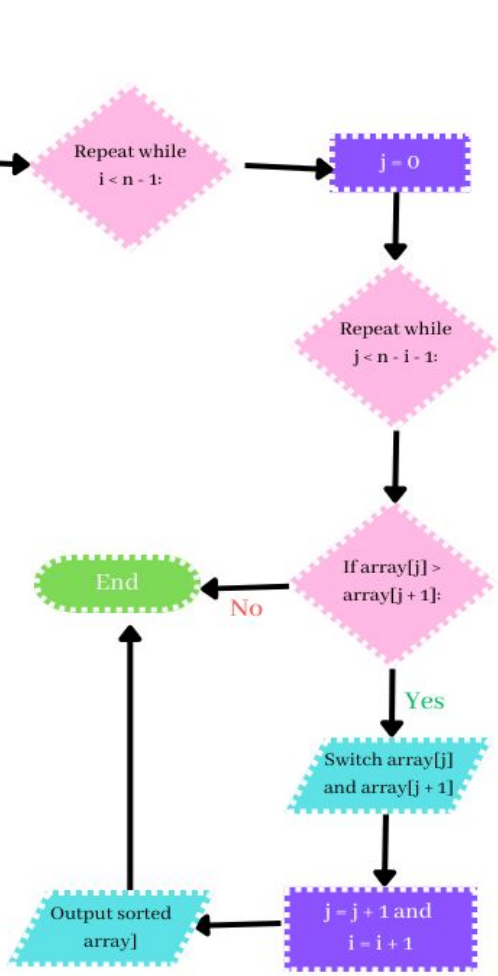
[1, 3, 4, 5, 11, 66, 84, 198]

Sorting an Array Pseudocode

```
- Define a function that takes an array as input and sorts it using bubble sort
def bubble_sort (array):
- Get the length of the array
n = len (array)
- Loop through the array n-1 times
for i in range (n-1):
- Loop through the array from 0 to n-i-1
for j in range (0, n-i-1):
- Compare the current element with the next element
if array[j] > array[j+1]:
- Switch them if they are out of order
array[j], array[j+1] = array[j+1], array[j]
- Return the sorted array
return array

- Test the function with an example
array = [1, 4, 5, 66, 3, 84, 11, 198]
sorted_array = bubble_sort (array)
print (sorted_array) # [1, 3, 4, 5, 11, 66, 84, 198]
```

Flow Chart


Start
Input Array
n = len(array)
i = 0
Repeat while i < n - 1:
j = 0
Repeat while j < n - i - 1:
If array[j] > array[j + 1]:
No
End
Yes
Switch array[j] and array[j + 1]
j = j + 1 and i = i + 1
Output sorted array]

Core: Search a number in a sorted matrix.

Challenge: You are given a matrix (a list of lists) of DISTINCT integers and a target number. Each row in the matrix is SORTED and each
column in the matrix is SORTED. Our matrix does not necessarily have the same height and width.

Write a pseudocode and produce a flowchart that:

Finds the number and report back its location (row and column indices of the target integer), if it is contained in the matrix

otherwise report back that the integer is not in the matrix.

EXAMPLE matrix:
matrix = [[1,4,7,12,15,1000], [2,5,19,31,32,1001], [3,8,24,33,35,1002], [40,41,42,44,45,1003], [99,100,103,106,128,1004]]

target = 44

EXAMPLE result:

result = [3,3]

Challenge Problem #3 (Core)

Note: Indexes start at 0

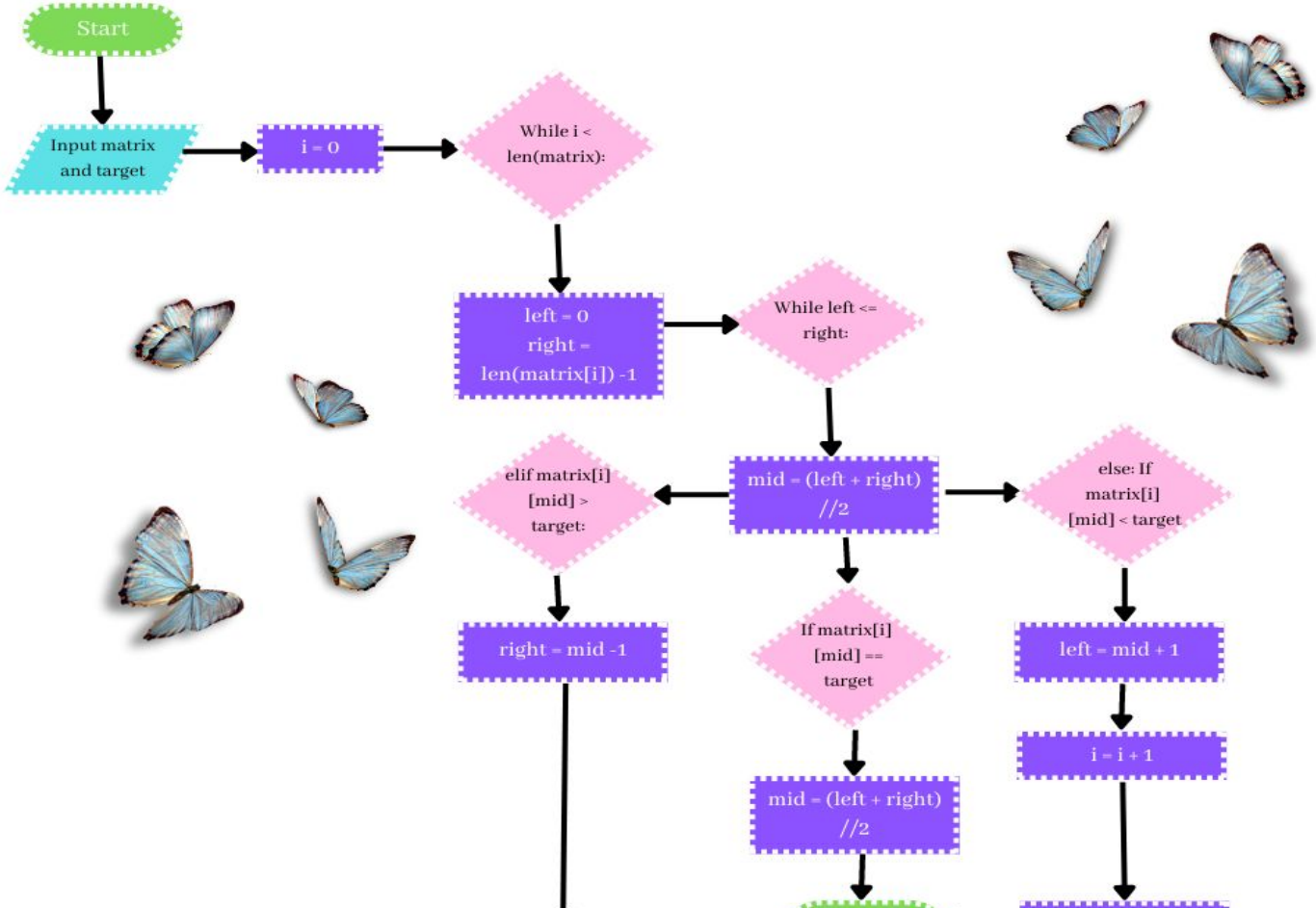
Integer 1 location is [0,0], integer 4 location is [0,1] and so on. The integers are

Search a Number in a Sorted Matrix Pseudocode

```
-Define a function to search a number in a sorted matrix
def search_matrix(matrix, target):
- Loop through each row of the matrix
for i in range(len(matrix)):
- Initialize the left and right pointers for binary search
left = 0
right = len(matrix[i]) - 1
- Repeat while left <= right
while left <= right:
-Calculate the middle index
mid = (left + right) // 2
- If the middle element is equal to the target, return its location
if matrix[i][mid] == target:
return [i, mid]
- If the middle element is greater than the target, move the right pointer to the left of the middle
elif matrix[i][mid] > target:
right = mid - 1
- If the middle element is less than the target, move the left pointer to the right of the middle
else:
left = mid + 1
- If the target is not found in any row, return -1
return -1
```

```
-Test the function with an example
matrix = [
[1,4,7,12,15,1000],
[2,5,19,31,32,1001],
[3,8,24,33,35,1002],
[40,41,42,44,45,1003],
[99,100,103,106,128,1004]
]
target = 44
result = search_matrix(matrix, target)
print(result) # [3, 3]
```

Flow Chart


Start
Input matrix and target
i = 0
While i < len(matrix):
left = 0
right = len(matrix[i]) -1
While left <= right:
mid = (left + right) //2
elif matrix[i] [mid] > target:
else: If matrix[i] [mid] < target
right = mid -1
If matrix[i] [mid] == target
left = mid + 1
i = i + 1
mid = (left + right) //2

Additional: Find factorial of n.

Challenge: The value of n will be small, less than 100. How could we use a lookup table to find the factorial not in the table already. If you would run the program, the first time the look up table would be empty.

Produce a pseudo code and a flowchart that allows us to find a factorial of a integer n.

EXAMPLE:

Factorial of 1 is 1! = 1.

Factorial of 2 is 2! = 1 * 2 = 2.

Factorial of 3 is 3!= 1 * 2 * 3 = 3 * 2! = 6

Factorial of 4 is 4!= 1 * 2 * 3 * 4 = 4 * 3! = 4 * 6

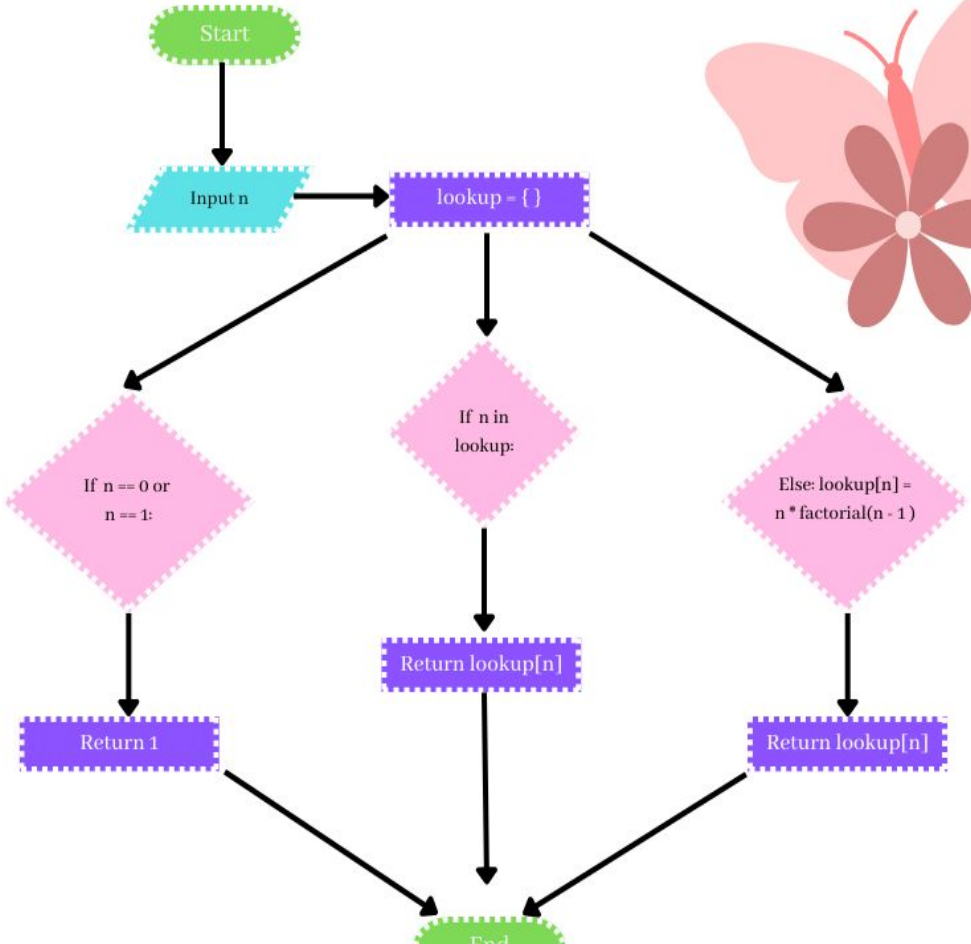
This gives us a general formula to find factorial: n! = n * (n-1)!

Find Factorial of n Pseudocode

- Define a function to find the factorial of n using a lookup table
def factorial(n):
-Initialize an empty dictionary to store the factorials
lookup = {}
- If n is 0 or 1, return 1
if n == 0 or n == 1:
return 1
- If n is in the lookup table, return its value
if n in lookup:
return lookup[n]
-Otherwise, calculate the factorial recursively and store it in the lookup table
else:
lookup[n] = n * factorial(n - 1)
return lookup[n]

-Test the function with some examples
print(factorial(1)) # 1
print(factorial(2)) # 2
print(factorial(3)) # 6
print(factorial(4)) # 24

Flow Chart


Start
Input n
lookup = { }
If n == 0 or n == 1:
If n in lookup:
Else: lookup[n] = n * factorial(n - 1)
Return 1
Return lookup[n]
Return lookup[n]

Additional: Hide the credit card digits.

Challenge: There are 16 digits on a credit card. Every 4 digits are separated by a space. Start by generating a random credit card number.

For security reasons, you are going to hide the first 12 digits on the credit card.

Example:

Randomly generated credit card number: 5486 3251 6584 7855

After hiding the first 12 digits, it would be: XXXX XXXX XXXX 7855

Produce a pseudo code and a flowchart.

Hide The Credit Card Digits Pseudocode

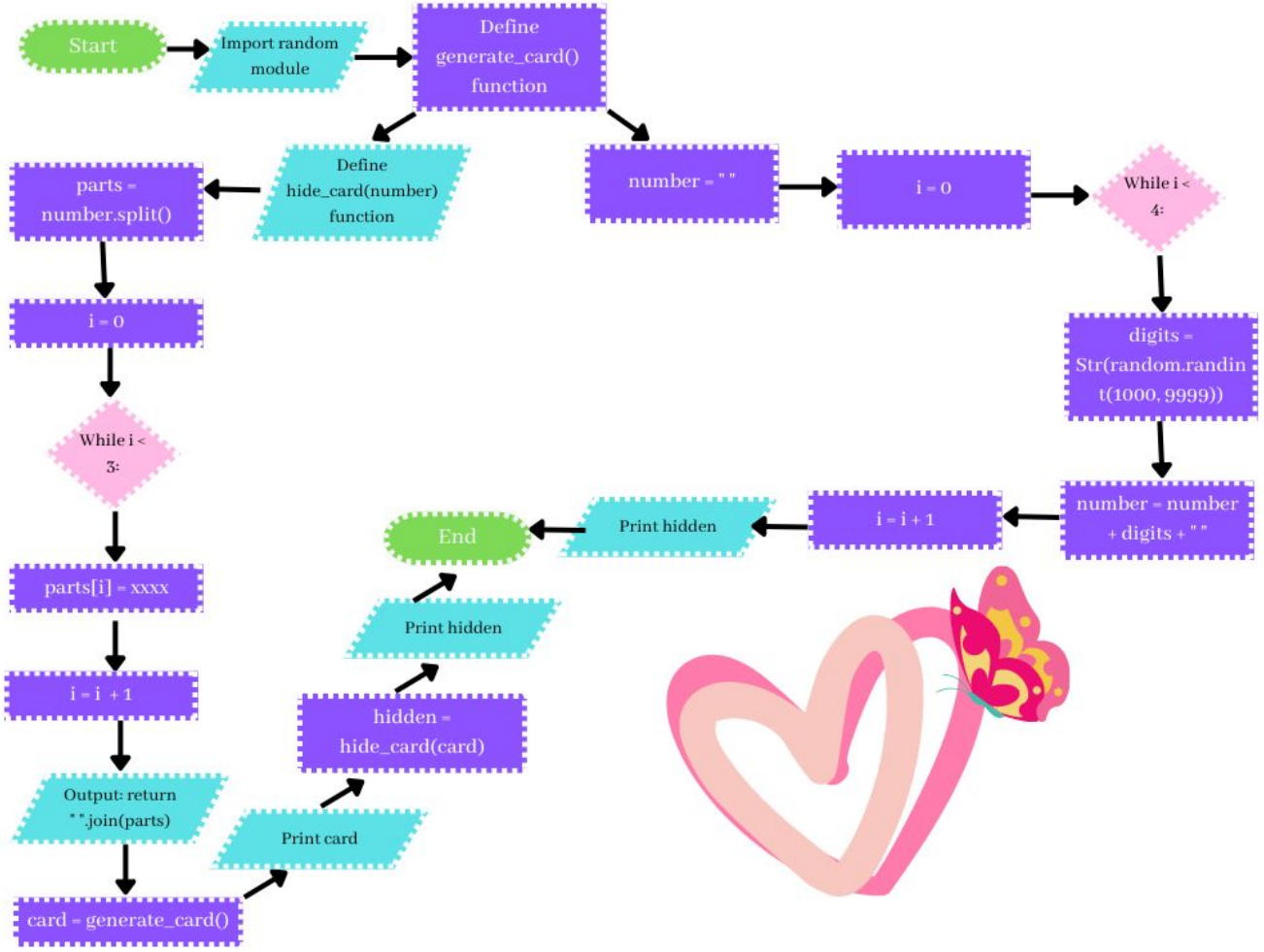
```
-Import the random module
import random

- Define a function to generate a random credit card number
def generate_card():
- Initialize an empty string to store the number
number = ""
- Loop 4 times
for i in range(4):
- Generate a random 4-digit number and convert it to a string
digits = str(random.randint(1000, 9999))
- Append the digits and a space to the number string
number = number + digits + " "
- Return the number string without the trailing space
return number.strip()

- Define a function to hide the first 12 digits of a credit card number
def hide_card(number):
                    -Split the number string by spaces and store it in a list
parts = number.split()
- Loop through the first 3 elements of the list
for i in range(3):
- Replace each element with "XXXX"
parts[i] = "XXXX"
-  Join the list elements with spaces and return the result
return " ".join(parts)

- Test the functions with an example
card = generate_card()
print(card)  # For example: 5486 3251 6584 7855
hidden = hide_card(card)
```

Flow Chart


Start
Import random module
Define generate_card() function
number = " "
i = 0
While i < 4:
digits = Str(random.randint(1000, 9999))
number = number + digits + " "
i = i + 1
Print hidden
End
Define hide_card(number) function
parts = number.split()
i = 0
While i < 3:
parts[i] = xxxx
i = i + 1
Output: return " ".join(parts)
card = generate_card()
Print card
hidden = hide_card(card)
Print hidden

FIN.

Thank You! 😊💞