



1. Write a program to create a 1D NumPy array of integers from 10 to 50 and print the array.

```
import numpy as np
a = []
for i in range (10,51):
    a.append(i)
b = np.array(a)
print(b)
```


 [10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50]

```
c = np.arange(10,51)
print(c)
```


 [10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50]

2. Write a program to create the following: a) A 2D array of zeros of size (3, 4) b) A 2D array of ones of size (4, 3) c) A 3x3 identity matrix.


```
zero_array = np.zeros((3,4))
print(zero_array)
```

 [[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]]

```
ones_array = np.ones((3,4))
print(np.full((3,4),6))
print(ones_array)
```


 [[6 6 6 6]
[6 6 6 6]
[6 6 6 6]]
[[1. 1. 1. 1.]
[1. 1. 1. 1.]
[1. 1. 1. 1.]]

```
identity = np.eye(3,dtype=np.int16)
print(identity)
```

 [[1 0 0]
[0 1 0]
[0 0 1]]


3. Create a NumPy array of 15 evenly spaced values between 0 and 5.

```
even_place_array = np.linspace(0,5,15)
print(even_place_array)
```

 [0. 0.35714286 0.71428571 1.07142857 1.42857143 1.78571429 2.14285714 2.5 2.85714286 3.21428571 3.57142857 3.92857143 4.28571429 4.64285714 5.]

4. Write a program to extract the second and third rows of the following array: arr = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])

```
arr = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])
extract_row = arr[1:3]
print(extract_row)
```

 [[40 50 60]
[70 80 90]]

5. Given an array arr = np.arange(10), write a program to reverse the array using slicing.

```
arr = np.arange(10)
reversed_arr = arr[::-1]
```

```
print(reversed_arr)
```

```
↩ [9 8 7 6 5 4 3 2 1 0]
```

6. Write a program to reshape the array `np.arange(12)` into a 3x4 matrix.

```
arr = np.arange(12).reshape(3,4)
print(arr)
```

```
↩ [[ 0  1  2  3]
   [ 4  5  6  7]
   [ 8  9 10 11]]
```

7. How do you flatten a 2D array into a 1D array? Demonstrate using an example.

```
array_2d = np.array([[12,13,14],[13,15,16],[12,16,81]])
flattend_array = array_2d.flatten()
print(flattend_array)
```

```
↩ [12 13 14 13 15 16 12 16 81]
```

8. Write a program to concatenate two NumPy arrays: `arr1 = np.array([1, 2, 3])`, `arr2 = np.array([4, 5, 6])` Concatenate them both row-wise and column-wise.

9. Write a program to calculate the element-wise sum, difference, product, and quotient of two arrays: `arr1 = np.array([1, 2, 3])`, `arr2 = np.array([4, 5, 6])`

```
arr1 = np.array([1,2,3])
arr2 = np.array([4,5,6])
row_wise = np.concatenate((arr1,arr2),axis=0)
print(row_wise)
col_wise = np.column_stack((arr1,arr2))
print(col_wise)
```

```
↩ [1 2 3 4 5 6]
   [[1 4]
    [2 5]
    [3 6]]
```

```
# Define the arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
```

```
# Element-wise operations
sum_result = arr1 + arr2
difference_result = arr1 - arr2
product_result = arr1 * arr2
quotient_result = arr1 / arr2
```

```
print(sum_result)
print(difference_result)
print(product_result)
print(quotient_result)
```

```
↩ [5 7 9]
   [-3 -3 -3]
   [ 4 10 18]
   [0.25 0.4 0.5 ]
```

10. Write a program to calculate the dot product of the following two arrays: `a = np.array([[1, 2], [3, 4]])`, `b = np.array([[5, 6], [7, 8]])`

```
# Define the arrays
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
```

```
# Calculate the dot product
```

```
dot_product = np.dot(a,b)
print(dot_product)
```

```
→ [[19 22]
    [43 50]]
```

11. Write a program to find the mean, median, and standard deviation of the following array: `arr = np.array([10, 20, 30, 40, 50])`

```
arr = np.array([10, 20, 30, 40, 50])
arr_mean = np.mean(arr)
arr_median = np.median(arr)
arr_std_deviation = np.std(arr)
print(arr_mean)
print(arr_median)
print(arr_std_deviation)
```

```
→ 30.0
   30.0
   14.142135623730951
```

12. Write a program to find the sum of all elements, rows, and columns of the given array: `arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])`

```
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
total_sum = np.sum(arr);
col_ele_sum = np.sum(arr,axis=0)
row_ele_sum = np.sum(arr,axis=1)
print(total_sum)
print(row_ele_sum)
print(col_ele_sum)
```

```
→ 45
   [ 6 15 24]
   [12 15 18]
```

13. Given the array `arr = np.array([10, 20, 30, 40, 50])`, write a program to filter out elements greater than 25.

```
arr = np.array([10, 20, 30, 40, 50])
filtered_arr = arr[arr>25]
print(filtered_arr)
```

```
→ [30 40 50]
```

14. Write a program to find the indices of all non-zero elements in the following array:

```
arr = np.array([0, 3, 0, 5, 6, 0, 7])
```

```
# Define the array
arr = np.array([0, 3, 0, 5, 6, 0, 7])

# Find the indices of non-zero elements
non_zero_indices = np.nonzero(arr)
```

```
# Print the result
print("Indices of non-zero elements:", non_zero_indices[0])
```

```
→ Indices of non-zero elements: [1 3 4 6]
```

15. Write a program to generate: a) A 1D array of 10 random integers between 0 and 100. b) A 2D array of shape (4, 4) with random floating-point numbers.

```
random_integers = np.random.randint(1,101,size=10)
print(random_integers)
random_floats = np.random.rand(4,4)
print(random_floats)
```

```
→ [89 80 18 31 53 74  2 95 69 65]
   [[0.69617488 0.40121997 0.96320937 0.47488552]
    [0.03006834 0.94732838 0.76987949 0.1595363 ]
```

```
[0.94058045 0.56656179 0.56937021 0.06615981]
[0.14385935 0.92583857 0.01523505 0.32289752]]
```

16. Write a program to set the random seed in NumPy and generate a 1D array of 5 random integers between 1 and 10. Show how the same output is reproduced if the seed is reset.

```
# Set the random seed and generate the array
np.random.seed(12)
array1 = np.random.randint(1, 11, size=5) # Random integers between 1 and 10
print("First array:", array1)

# Reset the seed and generate the array again
np.random.seed(13)
array2 = np.random.randint(1, 11, size=5) # Same integers due to the same seed
print("Second array (after resetting seed):", array2)

# Verify if the outputs are the same
print("Arrays are equal:", np.array_equal(array1, array2))
```

```
First array: [7 2 3 4 4]
Second array (after resetting seed): [3 1 1 7 3]
Arrays are equal: False
```

17. Given two arrays: `arr1 = np.array([1, 2, 3])`, `arr2 = np.array([4, 5, 6])` Write a program to: a) Stack them vertically into a 2D array. b) Stack them horizontally into a single 1D array.

```
# Given arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# a) Stack them vertically into a 2D array
vertical_stack = np.vstack((arr1, arr2))
print("Vertical stack:\n", vertical_stack)

# b) Stack them horizontally into a single 1D array
horizontal_stack = np.hstack((arr1, arr2))
print("Horizontal stack:", horizontal_stack)
```

```
Vertical stack:
[[1 2 3]
 [4 5 6]]
Horizontal stack: [1 2 3 4 5 6]
```

18. Write a program to generate a histogram with 10 bins for the following dataset: `data = np.random.randint(1, 101, size=50)`

```
import matplotlib.pyplot as plt

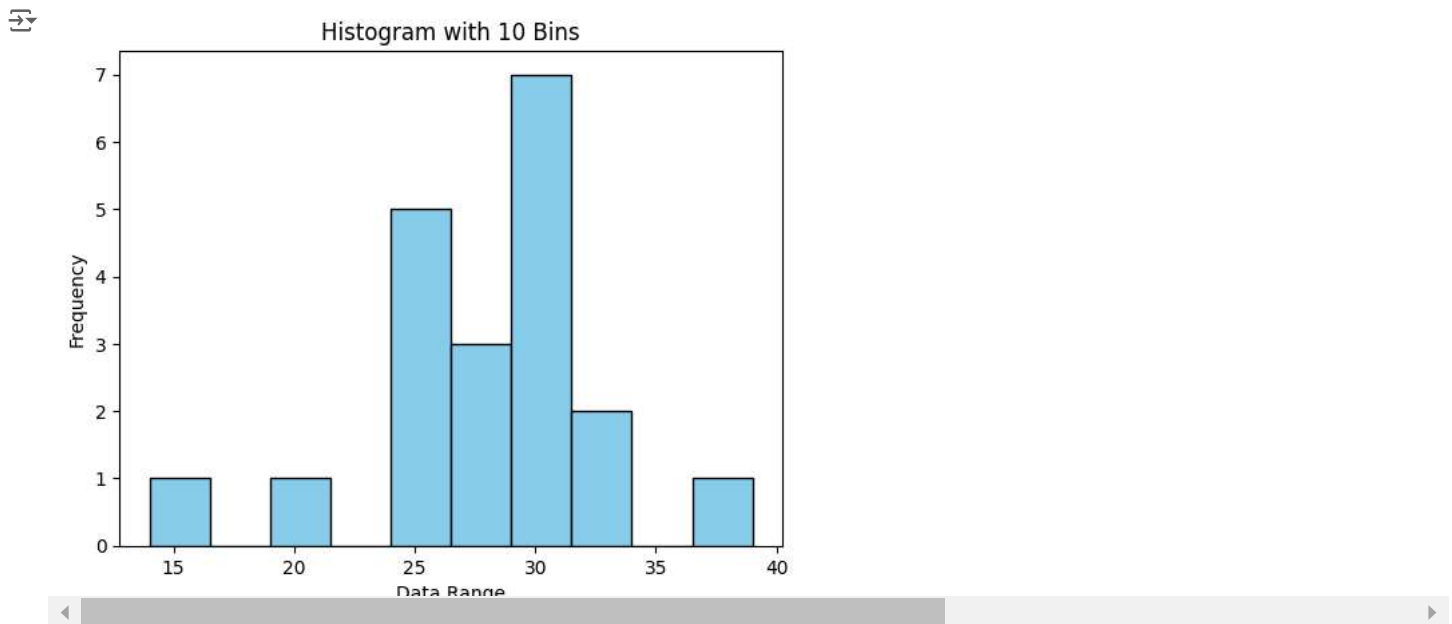
# Dataset
data = [30, 30, 30, 30, 30, 14, 26, 32, 25, 25, 25, 28, 32, 27, 39, 21, 25, 30, 30, 27]

# Number of bins
num_bins = 10

# Generate the histogram
plt.hist(data, bins=num_bins, edgecolor='black', color='skyblue')

# Add titles and labels
plt.title("Histogram with 10 Bins")
plt.xlabel("Data Range")
plt.ylabel("Frequency")

# Display the histogram
plt.show()
```



19. Write a program to simulate 20 experiments of flipping a fair coin 10 times each, using the binomial distribution. Print the number of successes (heads) in each experiment.

```
# Parameters
num_experiments = 20 # Number of experiments
num_flips = 10      # Number of coin flips per experiment
prob_head = 0.5     # Probability of heads

# Simulate experiments
results = np.random.binomial(n=num_flips, p=prob_head, size=num_experiments)

# Print the number of successes (heads) in each experiment
print("Number of successes (heads) in each experiment:")
for i, result in enumerate(results, start=1):
    print(f"Experiment {i}: {result} heads")
```

```
Number of successes (heads) in each experiment:
Experiment 1: 6 heads
Experiment 2: 6 heads
Experiment 3: 6 heads
Experiment 4: 6 heads
Experiment 5: 6 heads
Experiment 6: 5 heads
Experiment 7: 6 heads
Experiment 8: 6 heads
Experiment 9: 5 heads
Experiment 10: 5 heads
Experiment 11: 6 heads
Experiment 12: 3 heads
Experiment 13: 6 heads
Experiment 14: 5 heads
Experiment 15: 6 heads
Experiment 16: 5 heads
Experiment 17: 7 heads
Experiment 18: 4 heads
Experiment 19: 3 heads
Experiment 20: 7 heads
```

20. Use np.zeros, np.ones, mathematical operations and concatenation to create the following array: `array([[-1., -1., -1., -1.], [0., 0., 0., 0.], [2., 2., 2., 2.]])`.

```
# Create an array of zeros with shape (3, 4)
zeros_array = np.zeros((3, 4))

# Create an array of ones with shape (3, 4)
ones_array = np.ones((3, 4))

# Create the desired array using mathematical operations
```

```

negative_ones_array = -ones_array # This creates an array of -1s
positive_ones_array = ones_array * 2 # This creates an array of 2s

# Concatenate the arrays vertically
final_array = np.vstack([negative_ones_array, zeros_array, positive_ones_array])

print(final_array)

```

```

[[[-1. -1. -1. -1.]
  [-1. -1. -1. -1.]
  [-1. -1. -1. -1.]
  [ 0.  0.  0.  0.]
  [ 0.  0.  0.  0.]
  [ 0.  0.  0.  0.]
  [ 2.  2.  2.  2.]
  [ 2.  2.  2.  2.]
  [ 2.  2.  2.  2.]]

```

21. Create the following array: array([[2, 4, 6, 8, 10], [12, 14, 16, 18, 20], [22, 24, 26, 28, 30], [32, 34, 36, 38, 40]])

```

# Create an array with values from 2 to 40 with a step of 2
array = np.arange(2, 42, 2).reshape(4, 5)

print(array)

```

```

[[ 2  4  6  8 10]
 [12 14 16 18 20]
 [22 24 26 28 30]
 [32 34 36 38 40]]

```

22. Create matrix and access rows and columns a) create a 4x5 array of even numbers: 10, 12, 14, ... b) extract third column c) set the fourth row to 1,2,3,4,5

```

# a) Create a 4x5 array of even numbers: 10, 12, 14, ...
even_array = np.arange(10, 50, 2).reshape(4, 5)

print("Array of even numbers:")
print(even_array)

# b) Extract the third column
third_column = even_array[:, 2]

print("\nThird column:")
print(third_column)

# c) Set the fourth row to [1, 2, 3, 4, 5]
even_array[3] = [1, 2, 3, 4, 5]

print("\nUpdated array with fourth row set to [1, 2, 3, 4, 5]:")
print(even_array)

```

```

Array of even numbers:
[[10 12 14 16 18]
 [20 22 24 26 28]
 [30 32 34 36 38]
 [40 42 44 46 48]]

```

```

Third column:
[14 24 34 44]

```

```

Updated array with fourth row set to [1, 2, 3, 4, 5]:
[[10 12 14 16 18]
 [20 22 24 26 28]
 [30 32 34 36 38]
 [ 1  2  3  4  5]]

```

23. We can describe a coin toss as Binomial (1, 0.5) where 1 refers to the fact that we toss a single coin, and 0.5 means it has probability 0.5 to come heads up. So, such random variables are sequences of zeros and ones. But how can we get a sequence of -1 and 1 instead? Demonstrate it on com

```
# Number of coin tosses
num_tosses = 10

# Generate a binomial sequence (0 for tails, 1 for heads)
coin_tosses = np.random.binomial(1, 0.5, num_tosses)

# Transform the sequence to -1 and 1 (0 becomes -1, 1 stays as 1)
transformed_tosses = 2 * coin_tosses - 1

print("Original Coin Tosses (0 for tails, 1 for heads):", coin_tosses)
print("Transformed Tosses (-1 for tails, 1 for heads):", transformed_tosses)
```

```
→ Original Coin Tosses (0 for tails, 1 for heads): [0 1 0 0 1 0 1 1 0 1]
   Transformed Tosses (-1 for tails, 1 for heads): [-1  1 -1 -1  1 -1  1  1 -1  1]
```

24. consider two vectors `names = np.array(["Roxana", "Statira", "Roxana", "Statira", "Roxana"])` `score = np.array([126, 115, 130, 141, 132])`

Do the following using a single one-line vectorized operation. a) Extract all test scores that are smaller than 130 b) Extract all test scores by Statira c) Add 10 points to Roxana's scores. (You need to extract it first.)

```
names = np.array(["Roxana", "Statira", "Roxana", "Statira", "Roxana"])
score = np.array([126, 115, 130, 141, 132])
```

```
# a) Extract all test scores that are smaller than 130
smaller_than_130 = score[score < 130]
```

```
# b) Extract all test scores by Statira
statira_scores = score[names == "Statira"]
```

```
# c) Add 10 points to Roxana's scores
score[names == "Roxana"] += 10
```

```
print("Test scores smaller than 130:", smaller_than_130)
print("Test scores by Statira:", statira_scores)
print("Scores after adding 10 points to Roxana's scores:", score)
```

```
→ Test scores smaller than 130: [126 115]
   Test scores by Statira: [115 141]
   Scores after adding 10 points to Roxana's scores: [136 115 140 141 142]
```