

```

    [4, 2, 6, 1, 9],
    [3.0, -1.0, 10.0, -33.0],
    [1, 2, 3, 4, 5, 3],
    [5, 4, 3, 2, 1, 0],
]

for test_list in test_lists:
    print(f"Sorted {test_list} into {bubble_sort(test_list)}")

```


This program prints the following to the console:

```

Early return! Had 2 passes left.
Sorted [4, 2, 6, 1, 9] into [1, 2, 4, 6, 9]
Sorted [3.0, -1.0, 10.0, -33.0] into [-33.0, -1.0, 3.0, 10.0]
Early return! Had 4 passes left.
Sorted [1, 2, 3, 4, 5, 3] into [1, 2, 3, 3, 4, 5]
Sorted [5, 4, 3, 2, 1, 0] into [0, 1, 2, 3, 4, 5]

```

The lists are properly sorted. Note that in two cases the early return has occurred, as we can see by the early return messages. This demonstrates that more passes would have been performed without the early return logic.

Problem 2.6  **Preprogramming work:** In this problem, *before* writing the program specified, (1) draw a functional design method diagram (see ??) and (2) write a pseudocode for each function (see ??).

Restrictions: In this problem, most of the functions you will write already exist in the standard library module `statistics`. You may *not* use this module for this problem, but you may use others, such as the `math` module. You may also use list methods such as `sort()`. Furthermore, you may not use any external packages.

Programming: Write a program in a single script that meets the following requirements:

- a. It defines a function `stats(x: list) -> dict` that computes the following basic statistics for input list `x` of real numbers:
 - i. The sample mean; for a list `x` of n values, the sample mean m is

$$m(x) = \frac{1}{n} \sum_{i=0}^{n-1} x_i.$$

- ii. The sample variance; the sample variance s^2 is

$$s^2(x) = \frac{1}{n-1} \sum_{i=0}^{n-1} (x_i - m(x))^2.$$

iii. The sample standard deviation; the sample standard deviation s is

$$s(x) = \sqrt{s^2(x)}.$$


iv. The median; the median M of a *sorted* list x of n numbers is value of the list at index $i_M = (n - 1)/2$ (i.e., the middle index); more precisely,

$$M(x) = \begin{cases} x_{i_M} & i_M \text{ is an integer} \\ \frac{1}{2} (x_{\lfloor i_M \rfloor} + x_{\lceil i_M \rceil}) & \text{otherwise} \end{cases}$$

where $\lfloor \cdot \rfloor$ is the floor function that rounds down and $\lceil \cdot \rceil$ is the ceiling function that rounds up. So in the case that there is no middle index, the mode is the mean of the two middle values.

The `stats()` function should return a `dict` with the keys `"mean"`, `"var"`, `"std"`, and `"median"` correspond to values for the computed sample mean, variance, standard deviation, and median.

b. It demonstrates the `stats()` function works on three different lists of numbers.

Solution 2.6  The functional analysis is summarized in the block diagram of figure S2.2.

Algorithms 2 to 6 show pseudocode for each function.

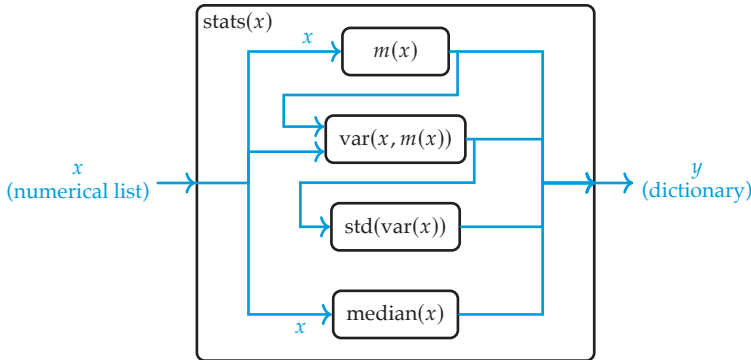


Figure S2.1. Functional analysis diagram.

Figure S2.2. Functional analysis diagram.

Algorithm 2 stats() pseudocode

```

function stats(x)
  mean_val ← mean(x)
  var_val ← var(x, mean_val)
  std_val ← std(var_val)
  med_val ← median(x)
  d ← dictionary of mean_val, var_val, std_val, med_val
  return d

```

Algorithm 3 mean() pseudocode

```

function mean(x)
  s ← sum(x)           ▷ Sum of elements
  n ← len(x)          ▷ Number of elements
  return s/n

```

Algorithm 4 var() pseudocode

```

function mean(x, m)
  if mean value m is not provided then
    m ← mean(x)
  summand ← []           ▷ Initialize new list
  for xi ∈ x do
    s ← (xi - m)2
    Append s to summand
  s ← sum(summand)
  n ← len(summand)
  return s/(n - 1)

```

Algorithm 5 std() pseudocode

```

function std(v)           ▷ v is the variance
  return √v

```

Algorithm 6 median() pseudocode

```

function median(x)
   $x' \leftarrow \text{sort}(x)$  ▷ New, sorted list
   $n \leftarrow \text{len}(x')$ 
   $i_M \leftarrow (n - 1) / 2$  ▷ Nominal middle index
  if  $i_M \in \mathbb{Z}$  (i.e., if it's an integer) then
    |  $med\_val \leftarrow x'_i$ 
  else
    |  $x_{\text{down}} \leftarrow x'_{\lfloor i_M \rfloor}$  ▷ Element from rounded-down index
    |  $x_{\text{up}} \leftarrow x'_{\lceil i_M \rceil}$  ▷ Element from rounded-up index
    |  $med\_val \leftarrow (x_{\text{down}} + x_{\text{up}}) / 2$  ▷ Mean
  return  $med\_val$ 

```

The following program meets the requirements:

```

"""Part of the Solution to Chapter 2 Problem YS"""
import math
from pprint import pprint

# %% [markdown]
## Introduction
# The program will consist of the five functions identified in the
# functional analysis diagram. The pseudocode for each function will
# guide the writing of the functions. The functions are defined in
# the following section and they are tested in the last section.
# %% [markdown]
## Function Definitions
# %%

def mean(x: list) -> float:
    """Returns the mean of a numeric list x."""
    s: float = sum(x) # Sum of items in list
    n: int = len(x) # Number of elements in list
    return s / n

def var(x: list, mean_val: float = None) -> float:
    """Returns the sample variance of x.
    Will compute the mean if it isn't supplied
    """
    if mean_val is None:
        mean_val = mean()
    summand = [] # Initialize the summand list
    for xi in x:
        summand.append((xi - mean_val) ** 2)
    s = sum(summand)
    n = len(summand)

```

```
    return s / (n - 1)

def std(var_val: float) -> float:
    """Computes the standard deviation from the variance."""
    return math.sqrt(var_val)

def median(x: list) -> float:
    """Returns the median of list x."""
    x_s: list = sorted(x) # New list
    n: int = len(x_s)
    i_m: float = (n - 1) / 2.0 # Nominal middle index
    if i_m.is_integer():
        med_val = x[int(i_m)] # Middle value
    else:
        x_low = x[math.floor(i_m)]
        x_high = x[math.ceil(i_m)]
        med_val = (x_low + x_high) / 2 # Mean of middle values
    return med_val

def stats(x: list) -> dict:
    """Returns a dict with the sample mean, variance,
    standard deviation, and median.
    """
    d = {}
    d["mean"] = mean(x)
    d["var"] = var(x, d["mean"])
    d["std"] = std(d["var"])
    d["median"] = median(x)
    return d

# %% [markdown]
## Call Functions and Print
# %%
test_lists = [
    list(range(0, 11)),
    [3.0, -1.0, 10.0, -33.0],
    [1, 2, 3, 4, 5, 3],
]

for test_list in test_lists:
    print(f"Stats for {test_list}:")
    pprint(stats(test_list), width=1)
```

This program prints the following to the console:

```
Stats for [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
{'mean': 5.0,
 'median': 5,
 'std': 3.3166247903554,
 'var': 11.0}
Stats for [3.0, -1.0, 10.0, -33.0]:
{'mean': -5.25,
 'median': 4.5,
 'std': 19.05037182489273,
 'var': 362.9166666666667}
Stats for [1, 2, 3, 4, 5, 3]:
{'mean': 3.0,
 'median': 3.5,
 'std': 1.4142135623730951,
 'var': 2.0}
```