Defining & Calling Functions

```
def calculate_cost(panels, price):
    total = panels * price
    return total

result = calculate_cost(12, 450)
print(f"Cost: ${result}")
```

return sends a value back. Without it, the function returns None.

Parameters & Arguments

```
def greet(name, greeting="Hello"):
    print(f"{greeting}, {name}!")

greet("Alice")              # Hello, Alice!
greet("Bob", "Good day")    # Good day, Bob!
```

Scope (Local vs Global)

Variables inside a function are **local** — they only exist inside it.

```
x = 10          # global

def my_func():
    x = 99      # local (different x!)
    print(x)    # 99

my_func()
print(x)        # 10 (unchanged)
```

Importing Modules

```
import math
print(math.sqrt(16))     # 4.0

from math import pi
print(pi)                # 3.14159...

import random as rnd
print(rnd.randint(1, 10))
```

Linear Search — O(n)

Check each element one by one. Works on **any** list.

```
def linear_search(items, target):
    for i in range(len(items)):
        if items[i] == target:
            return i    # found
    return -1           # not found
```

Partial matching (with in)

```
def search_panels(panels, term):
    results = []
    for panel in panels:
        if term.lower() in \
           panel["brand"].lower():
            results.append(panel)
    return results
```

Binary Search — O(log n)

Much faster but **requires sorted data**.

```
def binary_search(items, target):
    left = 0
    right = len(items) - 1

    while left <= right:
        mid = (left + right) // 2
        if items[mid] == target:
            return mid
        elif items[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1
```

Selection Sort — O(n²)

Find minimum, swap to front, repeat.

```
def selection_sort(items):
    n = len(items)
    for i in range(n - 1):
        min_idx = i
        for j in range(i + 1, n):
            if items[j] < items[min_idx]:
                min_idx = j
        items[i], items[min_idx] = \
            items[min_idx], items[i]
    return items
```

Quick Sort — O(n log n) average

Partition around a pivot, sort each half.

```
def quick_sort(items):
    if len(items) <= 1:
        return items
    pivot = items[len(items) // 2]
    left = [x for x in items if x < pivot]
    mid  = [x for x in items if x == pivot]
    right= [x for x in items if x > pivot]
    return quick_sort(left)+mid+quick_sort(right)
```

Algorithm Comparison

Algorithm	Time	Requires
Linear search	O(n)	Any list
Binary search	O(log n)	Sorted list
Selection sort	O(n²)	—
Quick sort	O(n log n)	—