Real-Time Example: Grading system calculation.

Python
```
score = 85

if score >= 90:
    grade = 'A'
elif score >= 80: # This runs only if score < 90
    grade = 'B'
elif score >= 70: # This runs only if score < 80
    grade = 'C'
else:
    grade = 'F'

print(f"Student score: {score}, Grade: {grade}") # Output: Grade: B
```

Nested `if` statements

Placing one `if` or `if-else` structure inside another `if` or `else` block. This allows for complex, multi-level decision-making.

Real-Time Example: E-commerce shipping qualification.

Python
```
is_prime_member = True
order_total = 75.50

if is_prime_member:
    if order_total >= 50:
        shipping_cost = 0.00
        print("🎉 Prime member with order over $50: Free shipping!")
    else:
        shipping_cost = 5.00
        print("Prime member, but order under $50. Shipping cost: $5.00")
else:
    # Non-Prime Member logic
    shipping_cost = 10.00
    print("Not a Prime member. Shipping cost: $10.00")
```

3.2 Looping Statements (Iteration)

Looping statements allow a block of code to be executed repeatedly.

The `while` loop

Executes a block of code **as long as** its condition remains `True`. You must ensure the condition eventually becomes `False` to avoid an infinite loop.

Real-Time Example: Countdown timer.

Python
```
timer = 5
```

```
print("Starting in...")
while timer > 0:
   print(timer)
   timer -= 1 # Crucial step to change the condition
   # In a real application, you'd use a time delay function here
print("GO!")
```

The for loop (Iterating over sequences/iterables)

Used for iterating over a sequence (like a list, tuple, dictionary, set, or string) or other iterable objects. It executes the code block once for each item in the sequence.

Real-Time Example: Processing items in a shopping cart.

Python

```
shopping_cart = ["milk", "bread", "eggs", "juice"]

print("Items in your cart:")
for item in shopping_cart:
   # 'item' takes the value of each element sequentially
   print(f"- {item.capitalize()}")
   # Real-time use: Calculate tax, update inventory, etc.
```

The range() function

Generates a sequence of numbers, often used to control for loops that need to run a specific number of times.

- range(stop): Generates numbers from 0 up to (but not including) stop.
- range(start, stop): Generates numbers from start up to (but not including) stop.
- range(start, stop, step): Generates numbers with the specified step size.

Real-Time Example: Displaying monthly sales data.

Python

```
# Iterating a fixed number of times (12 months)
monthly_sales = [1200, 1500, 1100, 1800, 2000, 2500, 2200, 1900, 1700, 1600, 2100, 2400]

print("Quarterly Sales Report:")
# i goes from 0 to 11 (12 months)
for i in range(len(monthly_sales)):
   # Using floor division to determine the quarter (0, 0, 0, 1, 1, 1, ...)
   quarter = (i // 3) + 1

   print(f"Month {i+1} (Q{quarter}): ${monthly_sales[i]}")

# Example using step: printing only odd numbers
print("\nOdd numbers from 1 to 10:")
for num in range(1, 11, 2):
   print(num, end=" ")
```

Nested loops

A loop inside another loop. The inner loop executes completely for every single iteration of the outer loop.

Real-Time Example: Generating a multiplication table or processing a 2D matrix (like seating charts).

Python
```
# Generating a small multiplication table
size = 3

print("\n--- Multiplication Table (3x3) ---")
for i in range(1, size + 1):  # Outer loop (rows)
    for j in range(1, size + 1):  # Inner loop (columns)
        # print(f"{i*j:4}", end="") # :4 is for padding
        print(i * j, end="\t") # \t is a tab space
    print() # Prints a newline after the inner loop finishes (end of row)
```

3.3 Loop Control Statements

These statements alter the normal execution flow of a loop.

break (Exiting the loop)

Immediately terminates the loop (both for and while) and transfers execution to the statement immediately following the loop.

Real-Time Example: Searching for a specific product in a large inventory.

Python
```
inventory = ["TV", "Phone", "Laptop", "Mouse", "Keyboard"]
target = "Laptop"

print("\nSearching inventory...")
for product in inventory:
    if product == target:
        print(f"🎯 Found the {target}! Stopping search.")
        break  # Exit the loop as soon as the item is found
    print(f"Checking {product}...")

# Code execution continues here, outside the loop.
print("Search complete.")
```

continue (Skipping the current iteration)

Stops the current iteration of the loop and moves the execution to the beginning of the next iteration.

Real-Time Example: Skipping weekend days when processing weekly tasks.

Python
```
working_days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]

print("\nProcessing tasks for the week:")
for day in working_days:
    if day == "Sat" or day == "Sun":
        print(f"Skipping {day}. Weekend!")
        continue  # Skips the rest of the code in the loop body for Sat/Sun
```

```
    # This code only runs for Mon-Fri
    print(f"Working on tasks for {day}...")
```

`pass` (Null operation statement)

The `pass` statement is a null operation; nothing happens when it executes. It is often used as a placeholder where a statement is syntactically required but you don't want any code to run.

Real-Time Example: Defining an incomplete function or condition block for later implementation.

Python
```
user_role = "admin"

if user_role == "admin":
    # Developer needs to add admin-specific code later
    pass
elif user_role == "user":
    print("Standard user functions loaded.")
else:
    # Handle unknown roles
    pass
```

`else` clause with loops (Executes if the loop finishes without a `break`)

The `else` block associated with a `for` loop executes *only if* the loop completes its full cycle (i.e., it was **not** terminated early by a `break` statement). This is most common in search operations.

Real-Time Example: Checking if an item is *not* found after checking the entire sequence.

Python
```
database = ["A101", "B202", "C303"]
search_id = "D404"

print("\nSearching database for ID D404...")
for item in database:
    if item == search_id:
        print(f"ID {search_id} found in database!")
        break
else:
    # This executes because the 'break' statement was NOT hit
    print(f"ID {search_id} was NOT found after checking all records.")
```