Sensor Networks Coverage Notes

Here's the content formatted as structured notes, classified into **Week 4**, **Week 5**, and **Week 6**:

Week 4: Sensor Networks – Part III

Target Tracking

- Nodes compute the target's position and notify the sink node periodically in a **push-based formulation** (uses cluster structure).
- In a **poll-based formulation**, nodes register the target's presence and send reports only when queried (uses tree structure).

Coverage

- Purpose: To collect relevant data for processing or reporting.
- Types of reporting:
  - **Event-driven:** e.g., forest fire monitoring.
  - **On-demand:** e.g., inventory control systems.
- Objective: Use minimum sensors and maximize network lifetime.
- Algorithms:
  - **Centralized:** Global map computed at a central point.
  - **Distributed:** Nodes compute positions using neighbor communication.
  - **Localized:** Only a subset of nodes participates in sensing, communication, and computation.
- Deployment: **Deterministic vs Random**.
- Considerations: **Sensing and communication ranges**.
- Static WSN coverage classifications:
  - **Area coverage**
  - **Point coverage**
  - **Barrier coverage**

Area Coverage

- Examples: **Energy-efficient random coverage**, **Connected random coverage**.
- A network is connected if any active node can communicate with another active node.

Barrier Coverage

- Types: **Weak coverage**, **Strong coverage**.

Coverage Maintenance

- A region R is covered if:
  - Crossings exist in R.
  - Every crossing is covered.
- **Crossings**: Intersection points between disk boundaries or between disk and region boundary.
- A crossing is covered if within at least one node's coverage disk.

Mobile Wireless Sensor Networks (MWSN)

- Intersection of WSNs and MANETs.
- Should follow **self-CHOP**:
  Self-Configure, Self-Heal, Self-Optimize, Self-Protect.
- WSNs have densely deployed sensor nodes that collaborate to measure environmental conditions.

Participatory Sensing

- Proposed by Burke et al. (2006).
- Distributed sensing by devices carried by humans.
- Goal: Data collection **and** knowledge sharing.
- Provides:
  - **Quantitative data** (e.g., CO_2 levels)
  - **Authenticity endorsement** via geo-tags and timestamps.

Gateway Selection in FANETs

- Begins with selecting the **most stable node** in a sub-area.

- Then, **partition parameters** are optimized based on topology.
- Metrics are optimized over iterations to reach an optimal state.

Week 5: Introduction to Arduino Programming

Device Interoperability

- System detects new devices, identifies capabilities, installs drivers/configs.
- Devices are controlled via command signals; status and sensor data are received.

Arduino IDE Overview

- Used to write and upload code to Arduino boards.
- Key functions:
  - **Verify**: Check for compilation errors.
  - **Upload**: Flash code to board (TX–RX LEDs flash).
  - **New**: Create new sketch.
  - **Open**: Open existing sketch (e.g., File → Examples).
  - **Save**: Save current sketch.
  - **Serial Monitor**: View printed data from the board.

Sketch Structure

- **setup()**: Runs once at start. Initializes I/O and pin modes.
- **loop()**: Runs continuously. Executes repetitive tasks.

Supported Datatypes

- Includes: `Void`, `Long`, `Int`, `Char`, `Boolean`, `Unsigned char`, `Byte`, `Unsigned int`, `Word`, `Unsigned long`, `Float`, `Double`, `Array`, `String` (char array and object), `Short`.

Example - Blink

- Connect board to PC.
- Select correct port and board type.
- Use:

- `digitalWrite(pin, value)` – Set pin to HIGH/LOW.
- `delay(ms)` – Pause for given milliseconds.

- Verify and upload code.

Operators in Arduino

- **Arithmetic:** `=`, `+`, `-`, `*`, `/`, `%`
- **Comparison:** `==`, `!=`, `<`, `>`, `<=`, `>=`
- **Boolean:** `&&`, `||`, `!`
- **Bitwise:** `&`, `|`, `^`, `~`, `<<`, `>>`
- **Compound:** `++`, `--`, `+=`, `-=`, `*=`, `/=`, `%=`, `|=`, `&=`

Control Statements

- **If**: Executes code if condition is true.
- **If...Else**: Executes different code blocks based on condition.
- **If...Elseif...Else**: Checks multiple conditions sequentially.
- **Switch Case**: Executes code based on variable's value.

Additional Topics Mentioned (Details Not Provided)

- **Loops**, **Arrays**, **String**, **Math Library**, **Random Number**, **Interrupts**, **Example Program**

Week 6: Introduction to Python Programming

Python IDE

- Free, open-source IDEs for coding and module/library integration.
- Available for Windows, Linux, Mac.
- Examples: **Spyder**, **PyCharm**.

Starting with Python

- Print using interpreter prompt:

```
python
```

```
>>> print "Hi, Welcome to python!"
```

- Python uses **rigid indentation** for blocks:

```python
if True:
    print "Correct"
else:
    print "Error"
```

Data-types in Python

- **Numbers**: Integers, floats, etc.
  Example: `x, y, z = 10, 10.2, " Python "`
- **String**: Sequence of characters.
  Indexing: `print x` → `T`
  Slicing: `print x[2:4]` → `is`
- **List**: Ordered, mutable. `x = [10, 10.2, 'python']`
- **Tuple**: Ordered, immutable.
- **Dictionary**: Key-value pairs. `d = {1:'item', 'k':2}`

Controlling Statements

- **if**, **elif**, **else**
- **while** loops with indented block after condition.

Functions in Python

- Define using `def`:

```python
def example(str):
    print (str + "!")
example("Hi")  # Output: Hi!
```

- Return multiple values:

```python
```

```python
def greater(x, y):
    if x > y: return x, y
    else: return y, x
val = greater(10, 100)
print(val)  # Output: (100, 10)
```

- Functions as objects:

```python
def add(a, b): return a + b
print(add(4, 6))
c = add(4, 6)
print c
```

File Read/Write Operations

- Steps: **Open**, **Read/Write**, **Close**
- Modes:
  - `'r'` – Read
  - `'w'` – Write (overwrites)
  - `'a'` – Append
  - `'r+'` – Read + Write
- Read:

```python
file = open('data.txt', 'r')
file.read()
```

- Write:

```python
file = open('data.txt', 'w')
file.write('writing to the file')
```

- Close:

```python
```

```
file.close()
```

- With block:

```python
with open("data.txt", "w") as file:
    file.write("writing to the text file")
```

CSV Files

- Use `csv` module.
- Functions: `csv.reader`, `csv.writer`.

Image Read/Write Operations

- Use **PIL (Pillow)** library.
- Install: `sudo pip install pillow`
- PIL for Python 2.7; Pillow supports Python 3.x.

Additional Note

- Python is one of the default installed languages on Raspberry Pi, indicating its importance in IoT.

Let me know if you want this in PDF or DOCX format too!