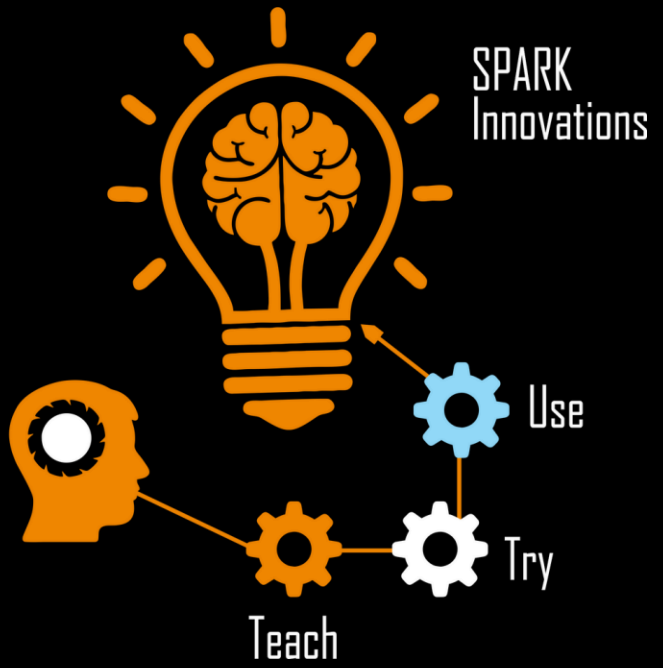
2026 SPARK ACADEMY

TRAIN FOR CHANGE, FROM SCIENCE TO PRACTICE

THE SPRINT AI TRAINING FOR AFRICAN MEDICAL
Imaging Knowledge Translation (SPARK) Academy
IN DEEP LEARNING & MEDICAL IMAGING

Week 2 Tutorial (Python Basics & Intro to NumPy)
FEBRAURY 28 2026



Today's Agenda

Week 2 — Building on the foundation

List Comprehensions

A faster, cleaner way to build lists

Functions

Write reusable blocks of code

Introduction to NumPy

Arrays, operations, slicing, 2D indexing

Medical Imaging Functions

Build SNR & CNR calculators

Quick Recap — Week 1

✓ Variables & Data Types

✓ Strings & f-Strings

✓ Lists & Dictionaries

✓ if/elif/else

✓ for & while Loops

✓ Built-in Functions

All of this is your foundation — today we build on it

Control Structures

Making decisions & repeating actions

if / elif / else — Deeper Practice

Make multiple decisions in sequence

```
# Classify patient priority
age = 72
tumor_cm = 3.5
is_emergency = True

if is_emergency:
    print("Priority: IMMEDIATE")
elif tumor_cm > 3.0 and age > 65:
    print("Priority: HIGH")
elif tumor_cm > 3.0:
    print("Priority: MEDIUM")
else:
    print("Priority: ROUTINE")
```

Output: `Priority: IMMEDIATE`

Key Points

Checks top to bottom

Stops at first True

Use 'and' / 'or' to

combine conditions

else catches everything

Nested Conditions

if inside another if — for complex decisions

```
# Determine if a scan is usable
scan_type = "MRI"
motion_artifact = False
snr_value = 25.0

if scan_type == "MRI":
    if motion_artifact:
        print("Rescan needed - motion detected")
    elif snr_value < 15:
        print("Low quality - consider rescan")
    else:
        print("Scan quality: GOOD")
else:
    print(f"Processing {scan_type} scan...")
```

Output: `Scan quality: GOOD`

for Loop Patterns

Common patterns you'll use constantly

```
# Pattern 1: Accumulator - build up a result
tumors = [1.2, 3.5, 0.8, 4.1, 2.9]
total = 0
for size in tumors:
    total += size
    avg = total / len(tumors)
    print(f"Average tumor size: {avg:.1f} cm")
```

Output: `Average tumor size: 2.5 cm`

```
# Pattern 2: enumerate - get index + value
scans = ["MRI", "CT", "X-Ray"]
for i, scan in enumerate(scans):
    print(f"Scan {i + 1}: {scan}")
```

Output:
```
Scan 1: MRI
Scan 2: CT
Scan 3: X-Ray
```

Loop Patterns

Accumulator
Sum, count, collect

`enumerate()`
Index + value together

`zip()`
Pair two lists together

while Loop — Repeat Until Done

```
# Simulate dose adjustment until target reached
current_dose = 10
target_dose = 50
step = 0

while current_dose < target_dose:
    current_dose += 10
    step += 1
    print(f"Step {step}: Dose = {current_dose} mGy")

print(f"Target reached in {step} steps!")
```

Output:
```
Step 1: Dose = 20 mGy
Step 2: Dose = 30 mGy
Step 3: Dose = 40 mGy
Step 4: Dose = 50 mGy
Target reached in 4 steps!
```

while vs for

for: know how many times to repeat

while: repeat until a condition is met

List Comprehensions

A faster way to build lists

List Comprehensions

Build a list in one line instead of 3-4 lines

Before (regular loop):

```
# Convert tumor sizes from cm to mm
sizes_cm = [2.3, 1.8, 4.1]
sizes_mm = []
for s in sizes_cm:
    sizes_mm.append(s * 10)
    print(sizes_mm)
```

After (list comprehension):

```
# Same thing in one line!
sizes_cm = [2.3, 1.8, 4.1]
sizes_mm = [s * 10 for s in sizes_cm]
print(sizes_mm)
```

Output: `[23.0, 18.0, 41.0]`

Syntax: [expression for item in list]

List Comprehension + Filter

Add a condition to filter items

```
# Get only critical tumors (> 3.0 cm)
tumors = [1.2, 3.5, 0.8, 4.1, 2.9]
critical = [t for t in tumors if t > 3.0]
print(critical)
```

Output: `[3.5, 4.1]`

```
# Get patient names with abnormal scans
patients = ["Amina", "Bola", "Chidi", "Dayo"]
results = ["Normal", "Abnormal", "Normal", "Abnormal"]

abnormal = [p for p, r in zip(patients, results)
            if r == "Abnormal"]
print(abnormal)
```

Output: `['Bola', 'Dayo']`

Syntax

```
[expr for item in list
        if condition]
```

The if at the end filters which items get included

Functions

Write once, use everywhere

Creating Functions

Reusable blocks of code with def

```
# Define a function
def classify_tumor(size_cm):
    if size_cm > 3.0:
        return "Critical"
    elif size_cm > 1.5:
        return "Monitor"
    else:
        return "Stable"

# Call it
print(classify_tumor(4.2))
print(classify_tumor(1.0))
```

Output:
```
Critical
Stable
```

Key Parts

def – defines it

parameters – inputs

return – output

call – use it

Functions with Multiple Parameters

```
def patient_report(name, age, tumor_cm):
    status = classify_tumor(tumor_cm)
    return f"{name} (age {age}): {tumor_cm} cm - {status}"

# Call it for different patients
print(patient_report("Fatima", 52, 3.5))
print(patient_report("Chidi", 34, 1.2))
```

Functions can call other functions!

Output:
```
Fatima (age 52): 3.5 cm - Critical
Chidi (age 34): 1.2 cm - Stable
```

```
# Default parameter values
def greet_patient(name, department="Radiology"):
    return f"Welcome {name} to {department}"

print(greet_patient("Amina"))
print(greet_patient("Kofi", "Oncology"))
```

Combining Functions

Build complex logic from simple pieces

```
# Process a batch of patients
def process_patients(patients):
    critical = []
    for p in patients:
        status = classify_tumor(p["tumor_cm"])
        if status == "Critical":
            critical.append(p["name"])
    return critical

patients = [
   {"name": "Fatima", "tumor_cm": 3.5},
   {"name": "Chidi", "tumor_cm": 1.2},
   {"name": "Amina", "tumor_cm": 4.1},
        ]
print(process_patients(patients))
```

Output: `['Fatima', 'Amina']`

Pattern

Small functions
↓
Combine them
↓
Solve big problems

Introduction to NumPy

The foundation of medical image processing

What is NumPy?

Numerical Python — fast math on arrays of numbers

```
import numpy as np

# A medical image is just a grid of numbers!
pixel_values = np.array([120, 145, 132, 158, 110])
print(pixel_values)
print(type(pixel_values))
```

Output: `[120 145 132 158 110]`
`<class 'numpy.ndarray'>`

Fast

100x faster than Python lists

Convenient

Built-in math operations

Foundation

Used by PyTorch, TensorFlow, nibabel

Creating NumPy Arrays

```
import numpy as np

# From a list
ages = np.array([45, 52, 34, 28, 61])

# Zeros and ones
blank_image = np.zeros((3, 3))
mask = np.ones((2, 2))

# Range and evenly spaced
slice_numbers = np.arange(0, 128)
dose_levels = np.linspace(0, 100, 5)
print(dose_levels)
```

Output: `[  0.  25.  50.  75. 100.]`

Common Creators

`np.array()`

`np.zeros()`

`np.ones()`

`np.arange()`

`np.linspace()`

`np.random.rand()`

Array Operations

Math on entire arrays at once — no loops needed!

```
import numpy as np

pixels = np.array([100, 150, 200, 50, 175])

# Arithmetic on every element
print(pixels + 10)     # Brightness up
print(pixels * 2)      # Double intensity
print(pixels / 255)    # Normalize to 0-1
```

Output:
```
[110 160 210  60 185]
[200 300 400 100 350]
[0.392 0.588 0.784 0.196 0.686]
```

```
# Operations between arrays
scan_a = np.array([100, 150, 200])
scan_b = np.array([90, 140, 210])
diff = scan_a - scan_b
print(diff)
```

No loops!

NumPy applies the operation to every element automatically. This is called vectorization.

NumPy Built-in Functions

```
import numpy as np

pixels = np.array([100, 150, 200, 50, 175])

print(np.min(pixels))    # 50
print(np.max(pixels))    # 200
print(np.mean(pixels))   # 135.0
print(np.std(pixels))    # 53.85
print(np.sum(pixels))    # 675
print(np.median(pixels)) # 150.0
```

`np.min()`
Minimum value

`np.std()`
Std deviation

`np.max()`
Maximum value

`np.sum()`
Total sum

`np.mean()`
Average

`np.median()`
Median value

These are essential for analyzing medical image data

Array Slicing & Indexing

Access parts of your data — just like lists but more powerful

```
import numpy as np

slice_data = np.array([10, 20, 30, 40, 50, 60, 70, 80])

print(slice_data[0])      # First: 10
print(slice_data[-1])     # Last: 80
print(slice_data[2:5])    # Slice: [30 40 50]
print(slice_data[:3])     # First 3: [10 20 30]
print(slice_data[::2])    # Every other: [10 30 50 70]
```

```
# Boolean indexing — very powerful!
pixels = np.array([50, 180, 30, 220, 90])
bright = pixels[pixels > 100]
print(bright)
```

Output: `[180 220]`

Boolean Indexing

Filter an array
using a condition.

Like SQL WHERE
but for arrays!

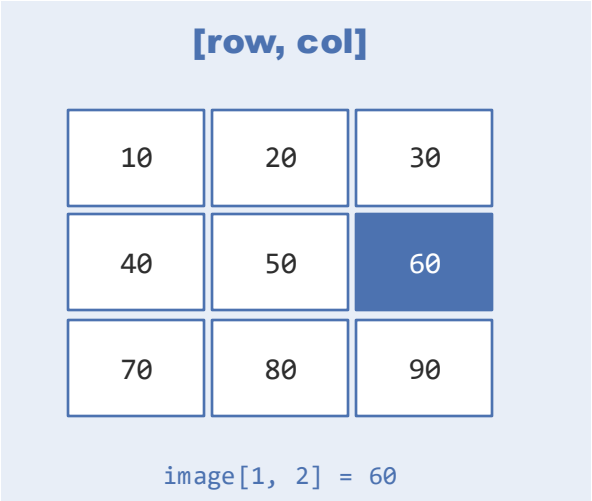
2D NumPy Arrays

Medical images are 2D (or 3D) arrays of pixel values

```
import numpy as np

# A tiny 3x3 "image"
image = np.array([
    [10, 20, 30],
    [40, 50, 60],
    [70, 80, 90]
])

print(image.shape)     # (3, 3)
print(image[0, 0])     # Top-left: 10
print(image[1, 2])     # Row 1, Col 2: 60
print(image[0, :])     # First row: [10 20 30]
print(image[:, 1])     # Second column: [20 50 80]
```



2D Slicing — Extracting Regions

Crop a region of interest from an image

```
import numpy as np

# Simulate a 5x5 brain scan slice
scan = np.array([
    [ 0,  0,  0,  0,  0],
    [ 0, 80, 90, 85,  0],
    [ 0, 95,200, 92,  0],
    [ 0, 82, 88, 80,  0],
    [ 0,  0,  0,  0,  0]
])

# Extract the 3x3 tumor region
tumor_region = scan[1:4, 1:4]
print(tumor_region)
print(f"Mean intensity: {np.mean(tumor_region):.1f}")
```

Output:
```
[[ 80  90  85]
 [ 95 200  92]
 [ 82  88  80]]
Mean intensity: 99.1
```

`scan[1:4, 1:4]`

Rows 1 to 3
Columns 1 to 3

This is how you crop a region of interest (ROI)!

Medical Imaging Functions

SNR & CNR — Your first real tools

Signal-to-Noise Ratio (SNR)

How clean is the signal compared to the noise?

$$SNR = \text{mean}(signal) / \text{std}(background)$$

```
import numpy as np

def calculate_snr(signal_region, background_region):
   """Calculate Signal-to-Noise Ratio"""
    signal_mean = np.mean(signal_region)
    noise_std = np.std(background_region)
    return signal_mean / noise_std

# Example: brain tissue vs background
signal = np.array([180, 195, 200, 185, 190])
background = np.array([12, 8, 15, 10, 11])

snr = calculate_snr(signal, background)
print(f"SNR: {snr:.2f}")
```

Output: `SNR: 78.33`

Contrast-to-Noise Ratio (CNR)

Can you tell two tissues apart in the image?

$$CNR = |mean(tissue\_A) - mean(tissue\_B)| / std(background)$$

```python
def calculate_cnr(tissue_a, tissue_b, background):
    """Calculate Contrast-to-Noise Ratio"""
    contrast = abs(np.mean(tissue_a) - np.mean(tissue_b))
    noise = np.std(background)
    return contrast / noise

tumor = np.array([200, 210, 195, 205])
healthy = np.array([120, 115, 125, 118])
background = np.array([12, 8, 15, 10])

cnr = calculate_cnr(tumor, healthy, background)
print(f"CNR: {cnr:.2f}")
```

Output:
```
CNR: 31.86
```

Putting It All Together

A complete image quality analysis

```
def image_quality_report(tumor, healthy, background):
    snr = calculate_snr(tumor, background)
    cnr = calculate_cnr(tumor, healthy, background)
    return {
       "snr": round(snr, 2),
       "cnr": round(cnr, 2),
       "tumor_mean": round(np.mean(tumor), 2),
       "healthy_mean": round(np.mean(healthy), 2),
       "noise_std": round(np.std(background), 2)
           }

report = image_quality_report(tumor, healthy, background)
for key, val in report.items():
    print(f"{key}: {val}")
```

Output:
```
snr: 78.33
cnr: 31.86
tumor_mean: 202.5
healthy_mean: 119.5
noise_std: 2.55
```

Keep Practicing!

Code Every Day!

Week 2 Foundation Assignment • Practice NumPy • Try the quiz

Thank You!

SPARK Academy 2026 — Week 2

Train for Change, From Science to Practice