AI in Digital Health Python Programming Course - Week 1
UoN MedTech

Digital Health

- Digital health encompasses various aspects of healthcare, including patient monitoring, prevention, treatment, administration, and research.
- Examples of digital health technologies include glucose monitors, smartwatches with ECG capabilities, and mental health apps.

AI in Digital Health

- AI is increasingly used in digital health for tasks such as prediction and analysis.
- The evolution of AI in healthcare has progressed from basic electronic health records to more complex applications like mobile health apps and AI-powered diagnostics.

Coding Fundamentals

- **Code** is a set of instructions that a computer can understand and execute.
- Programming languages are used to write code, allowing us to communicate with computers and give them instructions.
- Programming languages bridge the gap between human language and machine code (binary), making it easier to develop software and applications.
- **Syntax** refers to the set of rules that dictate how to write valid code in a programming language. It encompasses the correct structure, punctuation, and order of elements within a program.
- The **documentation** for a programming language and its many libraries (some of which we will get introduced to in future weeks) are the most reliable source to know what syntax to use. [Search google by typing ‘python docs’, followed by the name of the topic you want to know the syntax for. E.g. ‘for loop python docs’]

First Steps in Python Programming

- The `print()` function is used to display output in Python.
- To call a function, you type its name followed by parentheses containing any parameters.

Comments

- Comments are non-executable lines of code that provide explanations or prevent code from running.
- They are denoted by `#` or `##`.

- Comments are essential for code readability and understanding.

Inputs

- The `input()` function allows users to enter data, which is then stored in variables.
- Variables are used to store and manipulate data in a program.

Numerical Operations

- Python supports various numerical operations, including addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`), power (`^`), rounding (`round()`), remainder (`%`), and range (`in range()`).
- Data types define the characteristics of data and the operations that can be performed on them.
- Common data types include strings (`str()`), integers (`int()`), floats (`float()`), lists (`list`), tuples (`tuple()`), booleans (`bool()`), and None (`NoneType`).

List Operations

- Lists store ordered sequences of elements, enclosed in square brackets (``) and separated by commas.
- Lists are mutable, meaning their contents can be changed after creation.
- Indexing allows you to access individual elements in a list by their position, starting from 0.
- Slicing extracts a portion of a list, defined by a start and end index.
- The `len()` function returns the number of elements in a list.
- The `append()` method adds an element to the end of a list.

If Statements

- If statements execute a block of code only if a certain condition is true.
- Logical operators are used to create conditions, such as greater than (`>`), less than (`<`), equals (`=`), equals that returns True as a value (`==`), not equals (`!=`), less than or equal to (`<=`), and greater than or equal to (`>=`).

Sets

- Sets are unordered collections of unique elements, enclosed in curly braces (`{}`).
- Duplicates are automatically removed in sets.

Flow Control

- For loops iterate over a sequence of elements.
- Breaks (`break`) interrupt the execution of a loop.
- While loops repeat a block of code as long as a condition is true.
- Functions are reusable blocks of code that perform specific tasks.
- Functions are defined using the `def` keyword and can take parameters as input.

For Loops

- **Purpose:** For loops provide a way to iterate over sequences (like lists, tuples, or strings) and execute code for each element. They're particularly useful when you know exactly what collection of items you need to process.
- **Syntax:**

```python
for element in sequence:
    # Code to be executed for each element
```

- **Explanation:** The loop iterates through each element in the sequence, and the code within the loop is executed for each element.
- **Example:**

```python
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(f"I like eating {fruit}s")

# Output:
# I like eating apples
# I like eating bananas
# I like eating cherrys
```

While Loops

- **Purpose:** While loops repeatedly execute code as long as a specified condition remains true. They're ideal for situations where you don't know how many iterations will be needed.
- **Syntax:**

```python
while condition:
    # Code to be executed while the condition is true
```

- **Explanation:** The loop continues to execute the code within it until the condition becomes false.
- **Example:**

```python
count = 0
while count < 5:
    print(count)
    count += 1
```

This code will print the numbers from 0 to 4.

Key Differences

Feature	For Loop	While Loop
Use Case	**Known sequences**	**Conditional repetition**
Iterator	**Automatically handles iteration**	**Manual counter needed**
Termination	**Ends after sequence completion**	**Ends when condition is false**
Best For	**Finite collections**	**Unknown number of iterations**

Choosing Between For and While Loops

- Use a for loop when you need to iterate over a sequence of elements a known number of times.
- Use a while loop when you need to repeat a block of code until a certain condition is met, and the number of iterations is not known in advance.

Matrices

A **matrix** is a two-dimensional array of elements, commonly represented as a list of lists in Python. Each inner list represents a row of the matrix. You can access individual elements by using two indices: the first for the row and the second for the column.

Example of Matrix Operations in Python

Below is a quick example that demonstrates how to create a matrix, access its elements, and iterate over all its values.

Defining a Matrix

A 3x3 matrix can be defined as follows:

```python
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

Accessing Elements

To access a specific element in the matrix, use the row index followed by the column index. For example, to access the element in the second row, third column:

```python
element = matrix[1][2]  # This will be 6, as indexing starts at 0

print("Element at row 2, column 3:", element)
```

Iterating Over a Matrix

To iterate over the matrix and print each element:

```python
print("Matrix elements:")

for row in matrix:

    for value in row:

        print(value, end=" ")

    print()  # Newline after each row
```

This method prints all elements in a structured format, displaying each row separately.

If you want more explanations on these topics please see python's official tutorial: https://docs.python.org/3/tutorial/index.html