Artificial Intelligence with Python

Metropolia

1. Basics of Python Programming

- **Hello World**:

  ```
  print("Hello, World!")
  ```

- **Variables and Comments**:

  ```
  # This is a comment
  x = 10  # Variable assignment
  name = "Alice"
  print(name)
  ```

Metropolia

- **Indentation**: Python uses indentation instead of braces:

```
if x > 5:
    print("x is greater than 5")
```

- **Multi-line Strings**:

```
message = """This is a
multi-line string."""
print(message)
```

Metropolia

2. Operators in Python

- **Arithmetic Operators**:

```
x, y = 10, 3

print(x + y)   # Addition

print(x - y)   # Subtraction

print(x * y)   # Multiplication

print(x / y)   # Division

print(x // y) # Floor Division

print(x ** y) # Exponentiation

print(x % y)   # Modulus
```

Floor Division (//) always rounds down.Ceiling Division always rounds up.

Metropolia

- **Comparison Operators**:

```
print(x > y)    # Greater than

print(x == y)   # Equal to

print(x != y)   # Not equal to

print(x <= y)   # Less than or equal to
```

Metropolia

- **Logical Operators**:

```
print(x > 5 and y < 5)   # Logical AND

print(x > 5 or y > 5)    # Logical OR

print(not(x > y))        # Logical NOT
```

Metropolia

Logical Operators

Notation	Logical operator
and	"both"
or	"either" or "both"
not	negation, "no"

Let's assume that a and b are logical statements and their values are either true or false.

In that case:

statement a and b is true precisely if both statement a and statement b are true.

statement a or b is true when at least one of statements a and b are true.

statement not a is true precisely when statement a is false.

Metropolia

The order of precedence of the logical operators is as follows: the not operator is applied first, then the and operator and lastly the or operator. The order can be altered using parentheses.

Examples:

- statement a or b and c is true when either a is true or both b and c are true.
- statement (a or b) and c is true when at least one of statements a and b are true and also statement c is true.
- statement a and not b is true when a is true and b is false.

Metropolia

- **Membership Operators**:

```
fruits = ["apple", "banana", "cherry"]

print("apple" in fruits)  # True

print("grape" not in fruits)  # True
```

Metropolia

Data Types in Python

- **Immutable Data Types: : Numbers:**

```
x = 10   # Integer

y = 3.14   # Float

z = 2 + 3j   # Complex number

print(type(z))

#Mutable objects can be changed in place, without
creating a new object.

#Immutable objects cannot be changed, and any
modification results in a new object being
created.
```

Metropolia

Strings:

```
s = "Hello"
print(s[0])  # Access first character
print(s.upper())  # Convert to uppercase
```

Tuples:

```
coordinates = (10, 20, 30)
print(coordinates[1])  # Access second element
print(len(coordinates))  # Length of the tuple

# Tuples are immutable:
# coordinates[1] = 40  # This will raise an error
```

Tuples are like lists but immutable, meaning their values cannot be changed after creation. Useful for fixed data sets like coordinates.

Metropolia

Mutable Data Types:

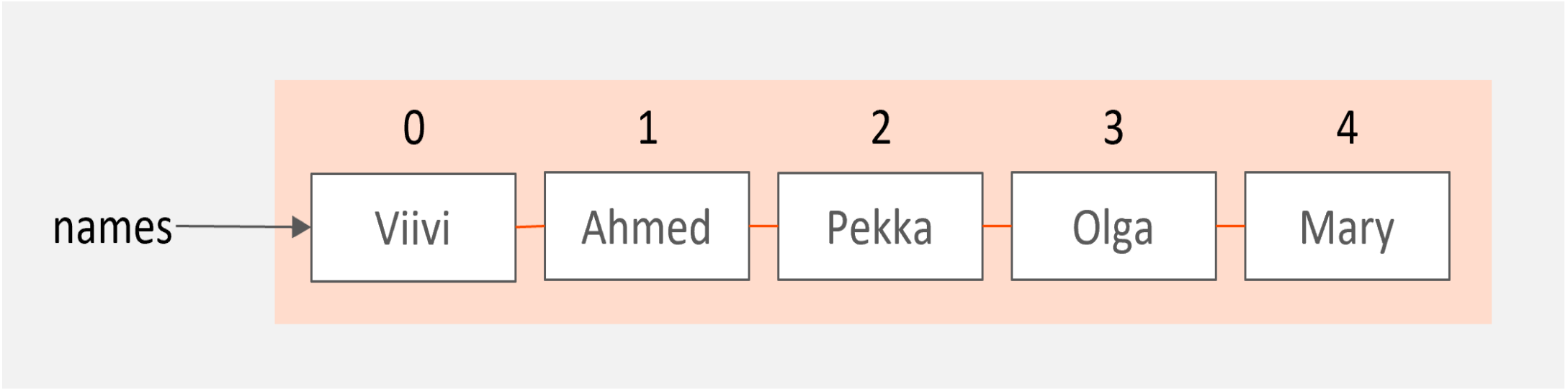
Lists:

```
fruits = ["apple", "banana", "cherry"]
fruits[1] = "blueberry"
fruits.append("date")
print(fruits)
```

Lists can grow, shrink, or have their elements modified.

Metropolia

List



Metropolia

Let's look at ways to reference list items. The following program prints out items and parts of a created list:

```
names = ["Viivi", "Ahmed", "Pekka", "Olga", "Mary"]

print(names[3])
print(names[1])
print(names[-2])
print(names[1:3])
print(names[2:])
print(names)
```

The output is as follows:

Metropolia

Olga
Ahmed
Olga
['Ahmed', 'Pekka']
['Pekka', 'Olga', 'Mary']
['Viivi', 'Ahmed', 'Pekka', 'Olga', 'Mary']

Metropolia

The most common list operations are listed in the table below:

Operation	Meaning	Example
append	adds an item to the end of the list	names.append("Matti")
remove	removes the first occurance of an item in the list	names.remove("Pekka")
insert	inserts an item into a defined position in the list, index specified in the first argument	names.insert(4, "Teppo")
extend	adds the items in the second list to the first list	otherNames = ["Allu","Ninni"] names.extend(otherNames)
index	returns the index of the first occurence of the specified item	what_index = names.index("Olga")
in	checks if an item exists in the list	if "Matti" in names: "Matti found"
sort	sorts the list items in alphabetical or numerical order	numbers.sort()

Metropolia

Dictionaries:

```
person = {"name": "Alice", "age": 25}
person["age"] = 26
person["city"] = "New York"
print(person)
```

Dictionaries store key-value pairs and allow modification by key.

Metropolia

Sets:

```
unique_numbers = {1, 2, 3, 4}
unique_numbers.add(5)
print(unique_numbers)
```

Sets hold unique items and are used for membership testing and eliminating duplicates.

Metropolia

4. Control Statements

- **if-else**:

```
x = 10
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")
```

Metropolia

for loop: The for loop is used for iterating over a **sequence** (like a list, tuple, or range) or other iterable objects.

The loop terminates automatically when the iterable is exhausted (e.g., when the range or list is fully traversed).

while loop : The while loop runs as long as a **condition** is True.

The loop terminates when the specified condition becomes False.

Metropolia

for Loop:

```
for i in range(1, 6):
    print(f"Iteration {i}")
```

while Loop:

```
count = 0
while count < 3:
    print(count)
    count += 1
rounds = int(input("How many greetings: "))
finished_rounds = 0
while finished_rounds<rounds:
    print("Good morning")
    finished_rounds = finished_rounds + 1
```

Metropolia

Functions:

Using functions helps you avoid situations where you would have to write and copy the same or a similar block of code to various parts of your program. Reusing the same code should always be avoided in programming as it makes programs more complex and more difficult to modify. If the reused code changes, the same change must be applied to more than one place in your program. Writing these repetitive tasks into functions instead solves this problem.

Functions are subroutines that are called from other parts of the program when needed.

Metropolia

```
# Function Definition
def calculate_rectangle_area(length, width):
    """This function calculates the area of a rectangle."""
    area = length * width
    return area

# Function Call
length = float(input("Enter the length of the rectangle: "))
width = float(input("Enter the width of the rectangle: "))
area = calculate_rectangle_area(length, width)  # Call the function with
  arguments

print(f"The area of the rectangle is: {area}")
```

Metropolia

6. Input and Formatting

- **Taking Input**:

```
name = input("Enter your name: ")
print(f"Hello, {name}!")
```

Metropolia

- **Formatted Strings**:

```
age = 25

height = 5.9

print(f"You are {age} years old and {height}
  feet tall.")
```

The f in the print(f"Iteration {i}") is part of an f-string (formatted string literal), which was introduced in Python 3.6. It allows you to embed expressions inside string literals using curly braces {}.

Metropolia

7. Exception Handling

The try and except blocks in Python are used for **exception handling**. They allow you to catch and handle errors (exceptions) that may occur during the execution of your code, instead of letting the program crash.

try Block

- The try block is used to wrap code that might raise an exception.
- If no exception occurs, the code in the try block is executed as normal.
- If an exception occurs, the rest of the code in the try block is skipped, and Python jumps to the except block.

except Block

- The except block is used to catch and handle exceptions raised in the try block.
- You can specify the type of exception you want to catch, or leave it general to catch all exceptions.
- Once an exception is caught, the code in the except block is executed.

finally block: Runs no matter what, useful for clean-up operations.

Metropolia

7. Exception Handling

- **Try-Except Block:**

```
try:
    num = int(input("Enter a number: "))
    print(10 / num)
except ValueError:
    print("Invalid input, please enter a number.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
```

Metropolia

Some commonly used built-in exceptions include:

- **ZeroDivisionError**: Raised when trying to divide by zero.
- **IndexError**: Raised when trying to access an index that is out of range in a list, tuple, or string.
- **ValueError**: Raised when a function receives an argument of the right type but inappropriate value (e.g., passing a non-numeric string to int()).
- **FileNotFoundError**: Raised when trying to open a file that doesn’t exist.
- **KeyError**: Raised when a dictionary key is not found.
- **TypeError**: Raised when an operation or function is applied to an object of inappropriate type.
- **AttributeError**: Raised when an invalid attribute reference is made, e.g., trying to access an attribute that doesn't exist.
- **NameError**: Raised when a local or global name is not found.

Metropolia

- **Finally Block**:

```
try:
    file = open("example.txt", "r")
except FileNotFoundError:
    print("File not found.")
finally:
    print("Execution completed.")
```

Exception handling makes programs robust by managing runtime errors.

The `finally` block is always executed.

Metropolia

Working with CSV Files:

Steps to Read and Display a CSV File

1. **Import pandas**: First, ensure the pandas library is installed in your Python environment.
2. **Read the CSV File**: Use the pandas.read_csv() function to load the CSV file into a DataFrame.
3. **Display Data**: Use the head() and tail() methods to preview the data.

Metropolia

Reading a CSV file:

```
import pandas as pd


# Read the CSV file into a pandas DataFrame
# Replace 'example.csv' with the path to your CSV file
df = pd.read_csv('example.csv’)
# Display the first 5 rows of the DataFrame
print("First 5 rows of the data:")
print(df.head())
# Display the last 5 rows of the DataFrame
print("\nLast 5 rows of the data:")
print(df.tail())
```

Metropolia

pd.read_csv('example.csv'):

- This reads the CSV file named example.csv and loads it into a pandas DataFrame.
- The DataFrame is a tabular data structure with labeled rows and columns.

1. **df.head()**:
   - Displays the first 5 rows of the DataFrame by default.
   - You can pass an integer n to view the first n rows (e.g., df.head(10)).
2. **df.tail()**:
   - Displays the last 5 rows of the DataFrame by default.
   - You can pass an integer n to view the last n rows (e.g., df.tail(3)).

Metropolia