AI in Digital Health
Python
Programming Course

UoN
MedTech

Overview of the Course

Overview of the Course

Weeks 1 & 2

- Focused on learning the basics of Python

Week 3

- Data analysis
- Real world health dataset

Week 4

- Introduction to Machine Learning and Deep learning

Weeks 5

- Deep Learning
- Computer Vision
- Natural Language Processing

Week 6

- Deep Learning (contd.)
- NLP with Transformers
- GPT
- Reinforcement Learning

Graduation Hackathon
15th March

Attendance and Certificates

There will be Attendance + Feedback QR codes at the end of each session

- **Feedback is optional**

Certificates will be given based on completion of these forms:

- **Certificate of Participation : Must attend 3 sessions excluding 1st session**

- **Certificate of Achievement : Must participate in the Hackathon**

Whatsapp Group QR

Support and Resources

Support

- **Weekly weekend Q&A sessions are available, or make use of the troubleshooting whatsapp group for questions**

Resources

- **Each week resources released**
  - Slides
  - Collab answers

Graduation Hackathon

Saturday 15th May

- A full-day event
- Teams come together to generate AI-driven healthcare ideas
- The goal is to create at least a UI mockup (or prototype if possible)
- At the end of the day, judges will review the projects and select the best ideas based on innovation, feasibility, and impact.

Week 3 - Data Analysis and Visualisation

More on Lists - Adding to lists

- `.append()` – Adds one item to the end of the list.
- `.extend()` – Adds multiple items to the end of the list from another group, like a list or a tuple (a fixed list).
- `.insert()` – Puts an item at a specific position in the list without replacing anything.

More on Lists - Removing from Lists

- `.remove()` – Deletes the first time a specific item appears in the list. If the item isn't found, it gives an error.
- `.pop()` – Removes an item by its position and also gives you that item back. If no position is given, it removes the last item.
- `.clear()` – Empties the list completely.

Getting information about lists and sorting

- `.index()` – Tells you where an item appears in the list for the first time. You can also choose to search only part of the list.
- `.count()` – Tells you how many times a specific item appears in the list.
- `.sort()` – Puts the list in order from smallest to largest (or in alphabetical order). You can also choose to sort it from largest to smallest.
- `sorted()` – Works like `.sort()`, but instead of changing the list, it creates a new one that's sorted.
- `.reverse()` – Flips the order of the list so the last item comes first and the first item goes last.
- `len()` – Tells you how many items are in the list.

- `.copy()` – Makes a new list with the same items as the original one, so changes to one don't affect the other.

List comprehensions

A Smarter Way to Create Lists in Python

```
squares = []
for x in range(10):
    squares.append(x**2)
squares
```

```
squares = [x**2 for x in range(10)]
squares
```

List comprehensions

Formula: [expression for item in list]

```
squares = []
for x in range(10):
    squares.append(x**2)
squares
```

```
squares = [x**2 for x in range(10)]
squares
```

List comprehensions - Adding a condition

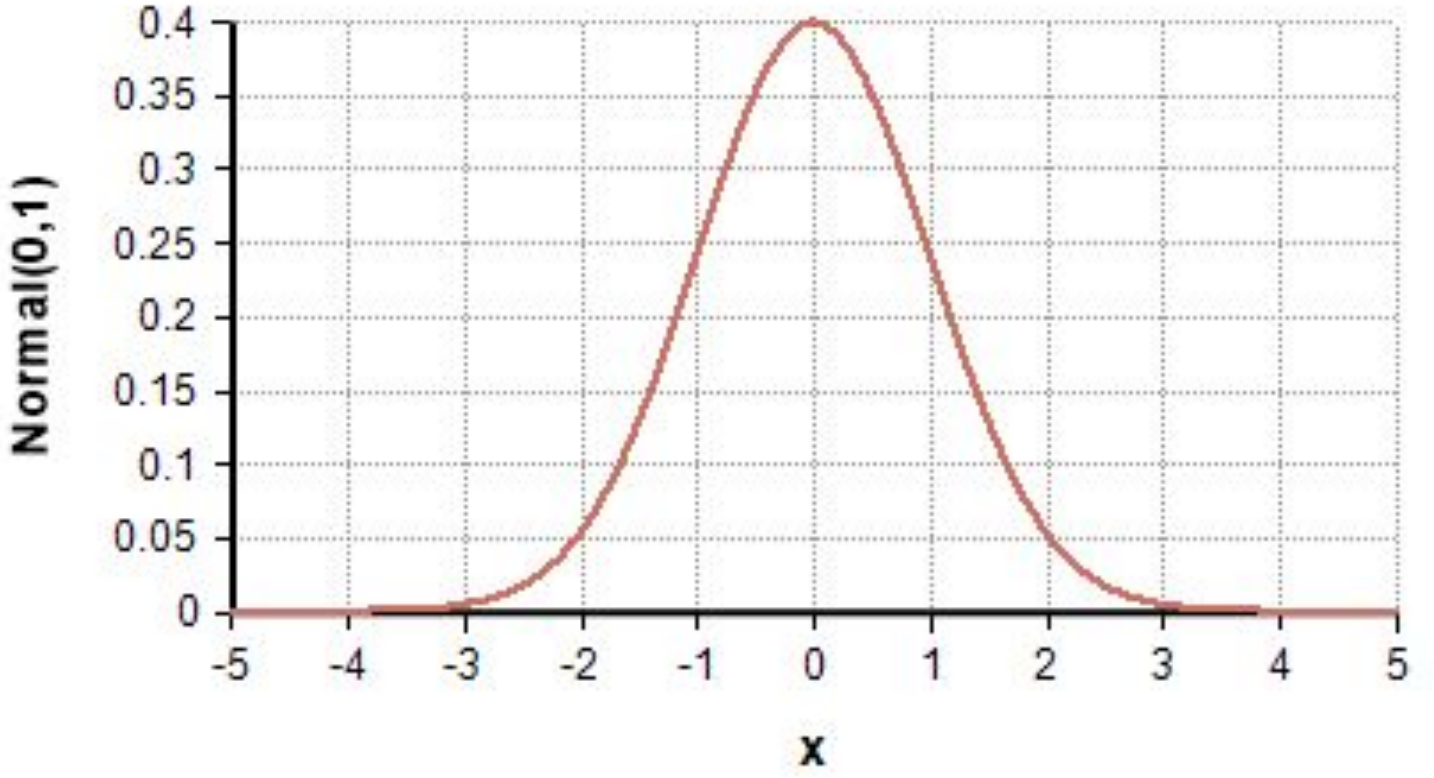
2 1 3

```
1  even_squares = [x**2 for x in range(10) if x % 2 == 0]
2  even_squares
```

List comprehensions - Random int

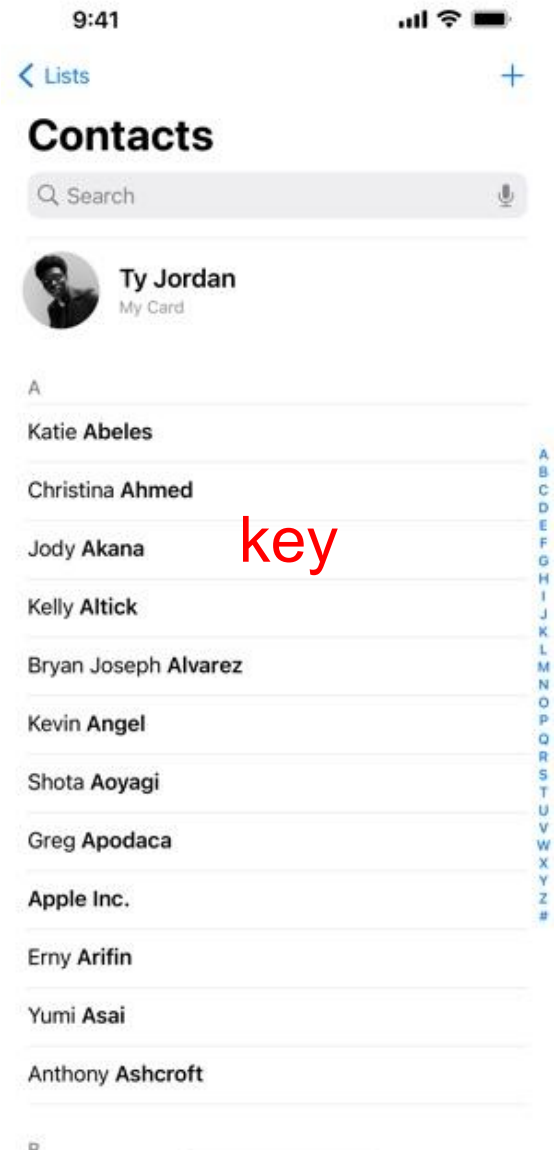

```
[random.randint(1, 100) for _ in range(10)]
```

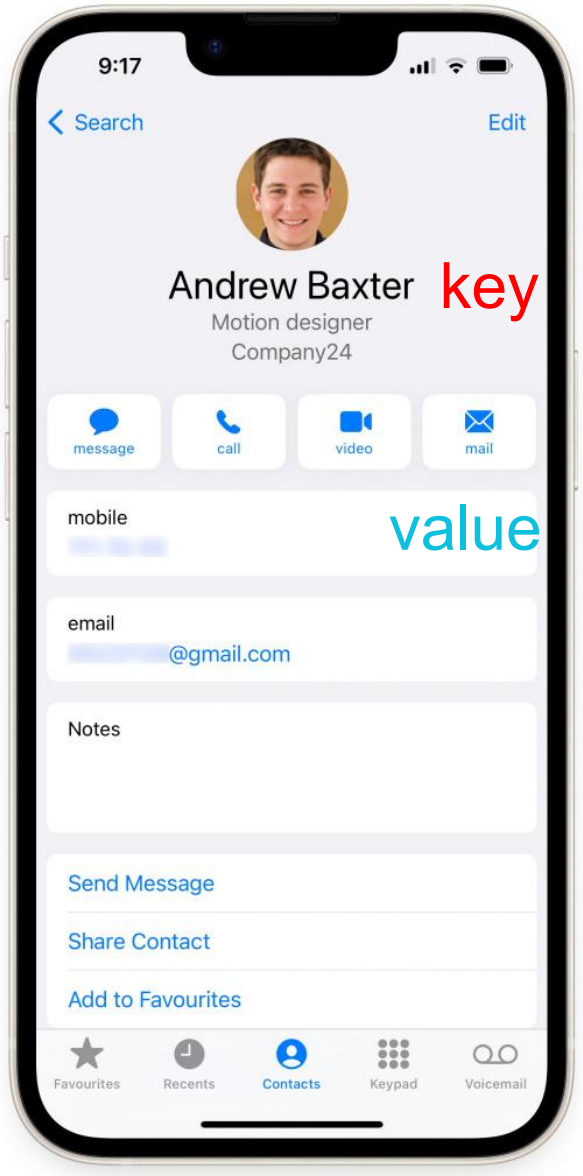
List comprehensions - Random gaussian




```
random_gaussians = [random.gauss(0, 1) for _ in range(10)]
```

Dictionaries

- 



Dictionaries

- 

A

key value

[1]a \'ā\ *n, pl* **a's** *or* **as** \'āz\ : 1st letter of the alphabet

[2]a \ə, 'ā\ *indefinite article* : one or some — used to indicate an unspecified or unidentified individual

aard·vark \'ärd,värk\ *n* : ant-eating African mammal

aback \ə'bak\ *adv* : by surprise

aba·cus \'abəkəs\ *n, pl* **aba·ci** \'abə,sī, -,kē\ *or* **aba·cus·es** : calculating instrument using rows of beads

abaft \ə'baft\ *adv* : toward or at the stern

ab·a·lo·ne \,abə'lōnē\ *n* : large edible shellfish

[1]aban·don \ə'bandən\ *vb* : give up without intent to reclaim — **aban·don·ment** *n*

[2]abandon *n* : thorough yielding to impulses

aban·doned \ə'bandənd\ *adj*

abate·ment \ə'bātmənt\ *n* : tax reduction

ab·at·toir \'abə,twär\ *n* : slaughterhouse

ab·bess \'abəs\ *n* : head of a convent

ab·bey \'abē\ *n, pl* **-beys** : monastery or convent

ab·bot \'abət\ *n* : head of a monastery

ab·bre·vi·ate \ə'brēvē,āt\ *vb* **-at·ed; -at·ing** : shorten — **ab·bre·vi·a·tion** \ə,brēvē'āshən\ *n*

ab·di·cate \'abdi,kāt\ *vb* **-cat·ed; -cat·ing** : renounce — **ab·di·ca·tion** \,abdi'kāshən\ *n*

ab·do·men \'abdəmən, ab'dōmən\ *n* **1** : body area between chest and pelvis **2** : hindmost part of an insect — **ab·dom·i·nal** \ab'dämən^{ə}l\ *adj* — **ab·dom·i·nal·ly** *adv*

ab·duct \ab'dəkt\ *vb* : kidnap — **ab·duc·tion** \-'dəkshən\

Dictionaries

```
one_person_data = {
    'Name': 'John Doe',
    'Age': 30,
    'Height': 175,
    'Weight': 70
}
```

Key → Unique identifier
Value → Stored information
Item → A full key-value pair

Dictionaries

Accessing Elements in a Dictionary - by key

```
# Accessing the value associated with the key 'Name'
patient_name = one_person_data['Name']
print(patient_name)
```

Dictionaries

Getting all keys

```
keys = dictionary.keys()
```

Dictionaries

Getting all Values

```
values = dictionary.values()
```

Dictionaries

Getting all Items

```
items = dictionary.items()
```

Checking if a Key Exists

```
'Age' in one_person_data
```

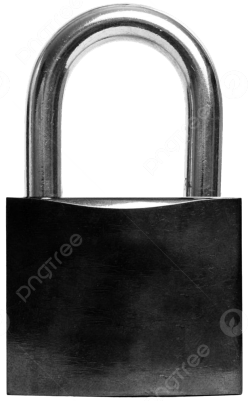
```
True
```

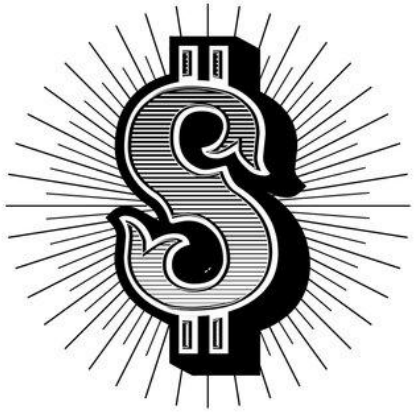
Safely access a dictionary

```
.get('key')
```

Generate Synthetic Dataset

Why? Real world data is:





Generate Synthetic Dataset

Names

```
1    fake = Faker()
2        return [fake.name() for _ in range(number_of_patients)]
```

Intro to Pandas & Numpy

Pandas → Works with tables (spreadsheet)

NumPy → Works with arrays (usually grids of numbers, can be other data types, but in in array all values must be of the same datatype**)**

Pandas - Create Dataframe

Dictionary -> Pandas dataframe (df)
.csv -> Pandas df

Pandas - Missing data

In the real world, a lot of datasets have missing data, so let's reflect that in our dataset

Pandas - look at data

3 quick ways:

`.head()` -> Shows the **first few rows** (default = 5).

`.info()` -> Gives a summary of columns, data types & missing values

`.describe()` -> Provides statistics for numerical columns.

Pandas - preprocessing

Preprocessing = Preparing (modifying) data before analysis.

Processing refers to actually analysing data to get new information.

We modify data to **clean, format, and structure** it.
Why? → Raw data is often messy or incomplete.

Pandas - preprocessing

Impute missing data

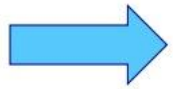
Imputation = Filling in missing data with a reasonable value.

One common method is mean imputation, where we replace missing values with the average of the column.

Mean imputation

$\bar{X}_2 = 3.8$
$\bar{X}_3 = 7.8$
$\bar{X}_4 = 2.7$

X_1	X_2	X_3	X_4
0	3	8	2
0	4	9	3
0	?	7	3
0	?	6	?
1	5	?	?
1	3	?	?
1	4	9	?



X_1	X_2	X_3	X_4
0	3	8	2
0	4	9	3
0	3.8	7	3
0	3.8	6	2.7
1	5	7.8	2.7
1	3	7.8	2.7
1	4	9	2.7

Pandas - preprocessing

Split Name into Firstname and Surname columns

`.str.split(' ', n=1, expand=True)` splits a column at the **first space**.

`'John Doe'` → `'John'` (Firstname), `'Doe'` (Surname).

Pandas - preprocessing

Problem:

- **Some names have titles** (e.g., *Dr. Lauren James)*
- The previous method **incorrectly assigns the title as the first name**.
- We need a smarter way to extract **Firstname** and **Surname**.

Pandas - preprocessing

Problem:

- **Some names have titles** (e.g., *Dr. Lauren James)*
- The previous method **incorrectly assigns the title as the first name**.
- We need a smarter way to extract **Firstname** and **Surname**.

Pandas - preprocessing

Solution:

```
def split_name(name):
    # Define common titles
    titles = ["Dr.", "Mr.", "Mrs.", "Ms.", "Miss", "Prof."]

    # Split the name into parts
    parts = name.split()

    # If first part is a title, remove it
    if parts[0] in titles:
        parts.pop(0)

    # First name = First word after title
    firstname = parts[0]

    # Surname = Everything else after firstname
    surname = " ".join(parts[1:]) if len(parts) > 1 else ""

    return pd.Series([firstname, surname])
```

Pandas - preprocessing

Feature engineering → Creating new data from existing data.

Name -> Gender

Pandas - preprocessing

Feature engineering → Creating new data from existing data.

Name -> Gender

1. **name_gender_df.groupby('Name') - Group data**
2. **.groupby('Name')['Gender'] - select the 'Gender' column from each group**
3. **.agg(lambda x: x.value_counts().idxmax()) - figure out which gender appears the most for each name**
4. **.to_dict() - After figuring out the most common gender for each name, we want to store this in an easy to lookup format, a dictionary.**

Graduation Hackathon

Saturday 15th May

- A full-day event
- Teams come together to generate AI-driven healthcare ideas
- The goal is to create at least a UI mockup (or prototype if possible)
- At the end of the day, judges will review the projects and select the best ideas based on innovation, feasibility, and impact.