Nosql databases

The Topic: What?

- An introduction to NoSQL Databases: document stores
    - A different type of database: Mongo DB Introduction
    - Using Mongo DB
    - Collections of Documents
    - Manipulating and Querying Documents

Applications: Why?

- To see an example of a NoSQL database – a document store that is not queried using SQL.
- To recognise how BSON documents and Python dictionaries can be used.

Expectations: Who?

- Learners are expected to have covered dictionaries in Python previously.

Persistent data

The Art of saving data to non-volatile storage

Data

Files

Database

Pickle

Shelve

Introduction to mongo db

MongoDB is a **document store**.
Document stores are *non-relational* databases - this means that each entry in the database (each 'document') represents a whole object rather than having objects spread across different tables.

Documents in Mongo are stored in BSON (Binary JavaScript Object Notation) which, like JavaScript Object Notation, is a system that involves storing data in key-value pairs.
For example, a person could be represented in BSON as:

```
{
    "_id" : ObjectId("5e135963c3782165689e6142"),
    "firstName": "Tadas",
    "secondName": "Vaidotas",
    "occupation": "Trainer",
    "age": 33,
    "specialisation": "DevOps"
}
```

Mongo database structure

Collections

Documents are grouped into collections based on their subject.
For example, a 'people' collection might be used to store documents like the previous example:

```
{
    "_id" : ObjectId("5e135963c3782165689e6142"),
    "firstName" : "Tadas",
    "lastName" : "Vaidotas",
    "age" : "33",
    "occupation" : "Trainer",
    "specialisation": "DevOps"
}
{
    "_id" : ObjectId("5e13599f9138596568d873ca"),
    "firstName" : "Jordan",
    "lastName" : "Harrison",
    "age" : "25",
    "occupation" : "Trainer"
}
```

While documents in a collection should have *something* in common, Mongo places no restriction on their structure.
Documents in a collection are flexible in composition - they might have identical fields, or basically nothing in common.

This leaves the composition of the data entered into that document entirely up to the person entering the data.
Notice how, in the example above, Tadas has a specialisation whereas Jordan does not.

Why?

Why is it used?

- Fits nicely into a JavaScript-based tech stack as the JSON-like format makes it very accessible to developers.
- Less restrictive as documents don't have to follow a pre-set schema.
- Immensely scalable through a process known as **sharding**.

When should you use it?

- When your data has no fixed structure.
- There exist no strong relationships between different collections.

Mongo db installation

Trainer demonstration

Windows

1.Download the windows binary from https://www.mongodb.com/download-center/community

2.Open Windows Explorer/File Explorer.

3.Change the directory path to where you downloaded the MongoDB .msi file. By default, this is %HOMEPATH%\Downloads.

4.Double-click the .msi file.

5.The Windows Installer guides you through the installation process.

Linux

Ubuntu

1.Update package repositories.

```
sudo apt update
```

2.Install via apt

```
sudo apt install mongodb -y
```

3. Check that the MongoDB server is active

```
systemctl status mongodb
```

Document store databases

A document store database simply acts as a store of collections.

Typically each database will be used by a different application.

This allows for the separation of collections in a Mongo instance based on what application requires them.

Create a database

The simplest way to create a new database is with the **use DB_NAME** command.

This command changes what database is currently selected but if the database you are trying to select does not already exist then it will be created before it is selected.
I.e., Creating an employee database:

```
use DB_NAME
```

Read

It is possible to show all the databases using

```
show dbs
```

This command only shows databases with *at least one collection*.

If you want to get the name of the currently selected database then it can be easily accessed via the **db** variable.

This variable always points to the name of the currently selected database and can therefore be used as a shortcut when trying to access collections of, run functions on, or perform any other operation on the current database.

Update

There is unfortunately no easy to rename a database in Mongo.

The simplest method is to copy your db, give the new db the name you want and remove the previous version, like so:

```
old_name.copyDatabase('old_name', 'new_name');
use old_name
old_name.dropDatabase();
```

Delete

Deleting a database can be done by selecting the database you want to delete and calling the **dropDatabase()** function.

```
use DATABASE_NAME
db.dropDatabase()
```

Create, display, and delete a mongo database

Trainer demonstration

1. Open a command window and connect to your Mongo instance.

```
mongo
```

2. Create a new database called **my_test_db** and select it.

```
use my_test_db
```

3. Display the current database.

```
db
```

4. Delete the newly created database.

```
db.dropDatabase()
```

Create a collection

To create a new collection in the current database simply use the **createCollection()** function, passing in the name of new collection.

```
db.createCollection('COLLECTION_NAME')
```

It is also possible to insert a document into a collection (or even a database) that does not exist yet.
Doing this will create the collection if it does not already exist.

It is possible to pass in an options object as an optional second argument.

This allows for the setting of a maximum memory amount or enforcing validation on documents added to the collection.

Read

It is possible to display all collections in the current database using the **show collections** command.

```
show collections
```

Update

If you want to rename a collection this can be accomplished using the **renameCollection()** function - simply call it on the collection you want to change and pass in the new name.

```
db.COLLECTION_NAME.renameCollection('NEW_NAME')
```

Delete

Deleting a collection can be done by calling the **drop()** function on the collection you no longer want.

```
db.COLLECTION_NAME.drop()
```

Create, display, and delete a mongo collection

Trainer demonstration

1. Create a new database.

```
use collections_tutorial
```

2. List the current collections - this should be blank right now.

```
show collections
```

3. Create a new collection.

```
db.createCollection('firstcollection')
```

4. Check the collection has been created.

```
show collections
```

Create, display, and delete a mongo collection

Trainer demonstration

5. Change the name of our new collection.

```
db.firstcollection.renameCollection('first_collection')
```

6. View the changed name.

```
show collections
```

7. Delete the new collection

```
db.first_collection.drop()
```

8. Check the collection has been deleted.

```
show collections
```

Create a document

Documents are how MongoDB stores data. Documents use a key-value pair format very similar to JSON files.

Documents do not have to follow a strict schema *unless* one has been set when the collection was created.

Insert a document by calling the **insertOne()** function on the collection you wish to add to and pass in the JSON object you want to store.

```
db.collectionName.insertOne({
    "firstName" : "Tadas",
    "lastName" : "Vaidotas",
    "age" : 33,
    "occupation" : "Trainer",
    "specialisation": "DevOps",
    "subjects": [
        "Docker",
        "AWS",
        "Scala"
    ]
})
```

Create a document

It is possible to insert more than one document at a time by passing a JSON array into the **insertMany()** function.

When documents are inserted into a Mongo database they are each given a **primary key** which acts as a unique identifier for each document.
In Mongo this primary key is the **_id** field.

```
db.collectionName.insertMany([
    {
        "firstName" : "Tadas",
        "lastName" : "Vaidotas",
        "age" : 33,
        "occupation" : "Trainer",
        "specialisation": "DevOps",
        "subjects": [
            "Docker",
            "AWS",
            "Scala"
        ]
    },
    {
        "firstName" : "Jordan",
        "lastName" : "Harrison",
        "age" : 25,
        "occupation" : "Trainer",
        "subjects": [
            "Java",
            "API dev",
            "Spring"
        ]
    }
])
```

Read

Pull all documents from a collection using the **find()** function.

```
db.collectionName.find()
```

It is possible to only pull certain documents by passing a **query** object into the **find()** function.

Projection

If you only want to find certain fields rather than the whole document then you can pass a **projection** object into the **find()** function, this will **project** out the fields we want into the query results.

```
db.practice.find(
    {},
    {
        "firstName": true,
        "lastName": true
    }
)
```

This query will find *all* documents (because we passed in a blank filter) but the only fields that will be displayed are **firstName**, **lastName** and **_id**.

Projection

This is because the **id** field will always be displayed unless specifically excluded, like so:

```
db.collectionName.find(
    {},
    {
        "_id": false
        "firstName": true,
        "lastName": true
    }
)
```

Note that normally it is only possible to include *or* exclude fields in a single projection but the **_id** field is an exception to this rule and can always be excluded.

Queries

It is possible to create very simple queries in Mongo by passing in partial objects.
For example, if you wanted to find someone with a first name of 'Tadas’

```
db.collectionName.find(
    {
        "firstName": "Tadas"
    }
)
```

For more complicated queries, you can use *query operators*.

Query operators

Equals

The **$eq** operator basically works in the same way as using no operator at all.
i.e., We could have written the previous example as

```
db.collectionName.find(
    {
        "firstName": { "$eq": "Tadas" }
    }
)
```

Obviously not that much point in using this operator so let's look at the not equals operator **$ne**.

```
db.collectionName.find(
    {
        "firstName": { "$ne": "Tadas" }
    }
)
```

This will flip the previous query and instead find every document where the firstName is *not* 'Tadas'.

Query operators

Greater/Less than

For numerical fields it is possible to find all docs where a value is *greater than* a particular number using **$gt**.
For example, finding all trainers over the age of thirty:

```
db.collectionName.find(
    {
        "age": { "$gt": 30 }
    }
)
```

You can specify *greater than or equal to* conditions using **$gte**.
This will find any trainers that are *at least* 30:

```
db.collectionName.find(
    {
        "age": { "$gte": 30 }
    }
)
```

Query operators

Greater/Less than (cont.)

Like wise we can do *less than* with **$lt**:

```
db.collectionName.find(
    {
        "age": { "$lt": 30 }
    }
)
```

And *less than or equal to* with **$lte**:

```
db.collectionName.find(
    {
        "age": { "$lte": 30 }
    }
)
```

Query operators

In/Nin

```
db.collectionName.find(
    {
        "specialisation": {
                        "$in": [
                            "Software Dev",
                            "DevOps"
                            ]
                        }
    }
)
```

```
db.collectionName.find(
    {
        "specialisation": {
                        "$nin": [
                            "Software Dev",
                            "DevOps"
                            ]
                        }
    }
)
```

Update

To change existing documents you will need to use **update query operators**, for example if I wanted to give Jordan a specialisation I could use the **updateOne()** function, passing in a filter and an update.

```
db.collectionName.updateOne(
    {
    "firstName": "Jordan",
    "lastName": "Harrison"
    },
    {
        "$set" : {
            "specialisation": "Software Development"
            }
    }
)
```

Note the use of the **$set** operator.
In MongoDB **$** is used to signify system fields or operators - in this case it is used to differentiate between the set **operator** and just a field called set.

Update

It is possible to update multiple documents in a similar manner using the **updateMany()** function.
The only real difference
between **updateMany** and **updateOne** is
that **updateOne** stops looking after it finds any document that matches the filter,
whereas **updateMany** will search the whole collection looking for any matching documents.

```
db.collectionName.updateMany(
    {},
    {
        "$set": {
            "reportsTo": "John Gordon"
        }
    }
)
```

Update

By passing in an empty filter the update operation will target *every* document in the collection and add the new field "reportsTo" with a value of "John Gordon".

If you want to replace an existing document with a brand-new one rather than updating an existing doc you can use the **replaceOne()** function.

```
db.collectionName.replaceOne()
```

Arrays

Values can be added into arrays using the **$push** operator

```
db.collectionName.updateOne(
    {
        "firstName": "Jordan",
        "lastName": "Harrison"
    },
    {
        "$push": {
            "subjects": "MongoDB"
        }
    }
)
```

Arrays

And similarly removed using the **$pull** operator.

```
db.collectionName.updateOne(
    {
        "firstName": "Tadas",
        "lastName": "Vaidotas"
    },
    {
        "$pull": {
            "subjects": { "$nin": "scala"}
        }
    }
)
```

Delete

The delete functions work in much the same way as the update functions except without the need for update operators.
For example, if you want to delete one document:

```
db.collectionName.deleteOne(
    {
        "firstName": "Jordan"
    }
)
```

This will delete the first document found where the firstName field has a value of Jordan.
Similarly you can delete *all* the documents that meet a certain criteria:

```
db.collectionName.deleteMany(
    {
        "specialisation": "DevOps"
    }
)
```

This command will delete any trainer with a DevOps specialisation.

Embedded documents

Through the power of JavaScript it is possible to put a document *inside* of another document.
This is typically used to represent an entity that belongs to another entity - for example, we might represent a person's job as a separate entity *embedded* inside the person entity.

```
{
    "firstName": "Jordan",
    "surname": "Harrison",
    "age": 25,
    "height": 182,
    "hobbies":  [
        "Gaming",
        "Reading",
        "Writing course-ware"
    ],
    "job": {
        "title": "Learning specialist",
        "salary": 1000000000,
        "startDate": new Date("2018-09-24"),
        "manager": "Christopher Perrins"
    }
}
```

Learning check

5-10 mins

Quiz!

1. **What command makes a database?**
2. **What command allows you to create a collection?**
3. **What command allows you to insert a single document?**
4. **How do you query a Mongo DB database?**

Solutions

NoSQL Databases Quiz

Quiz!

1. What command makes a database?

Use nameOfDatabase

2. What command allows you to create a collection?

db.createCollection(‘nameOfCollection’)

3. What command allows you to insert a single document?

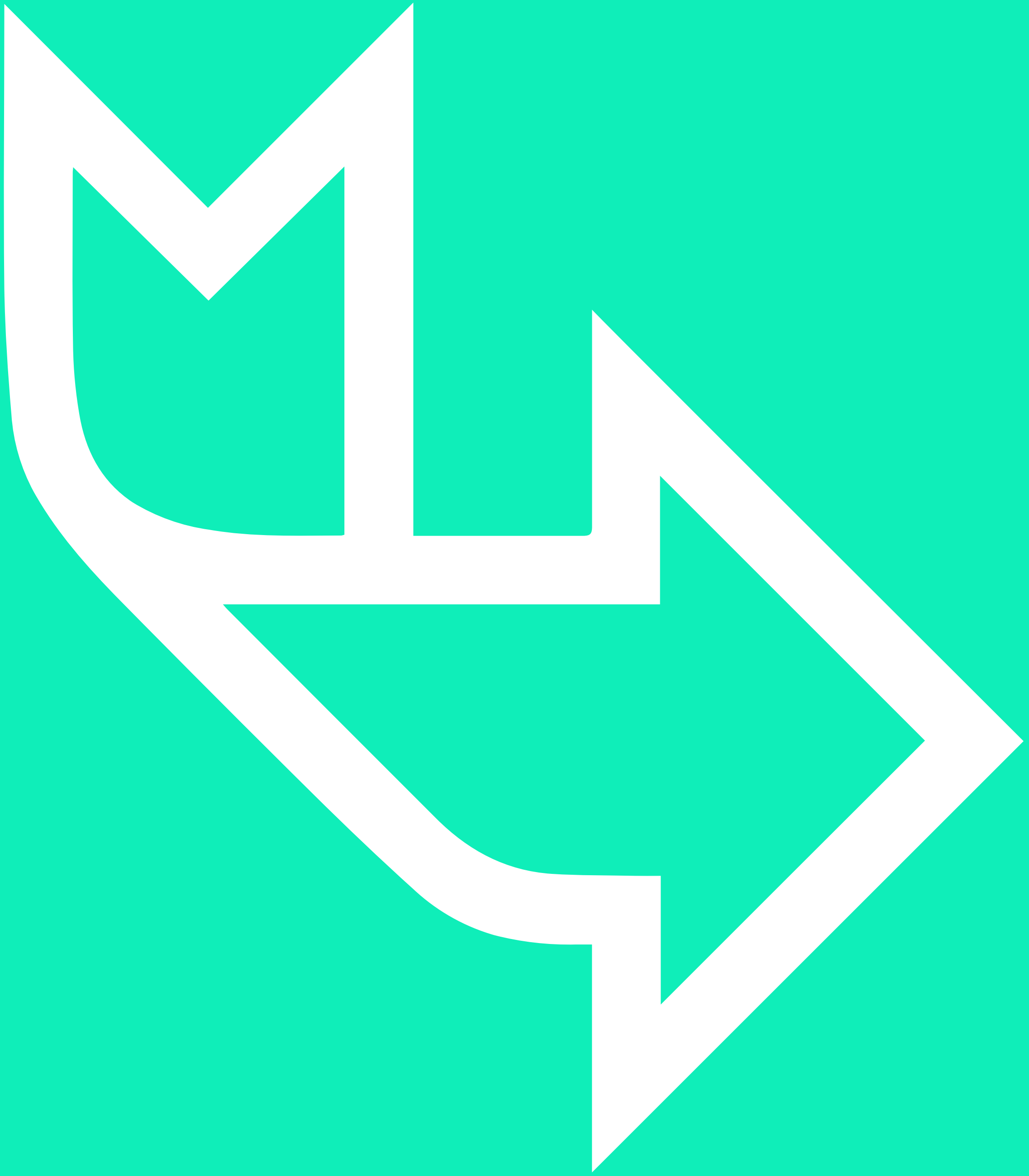
db.nameOfCollection.insertOne()

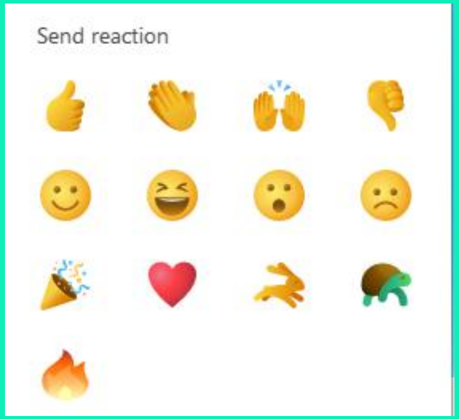
4. How do you query a Mongo DB database?

db.nameOfCollection.find()

Labs

Please refer to the lab instructions.





END OF SECTION

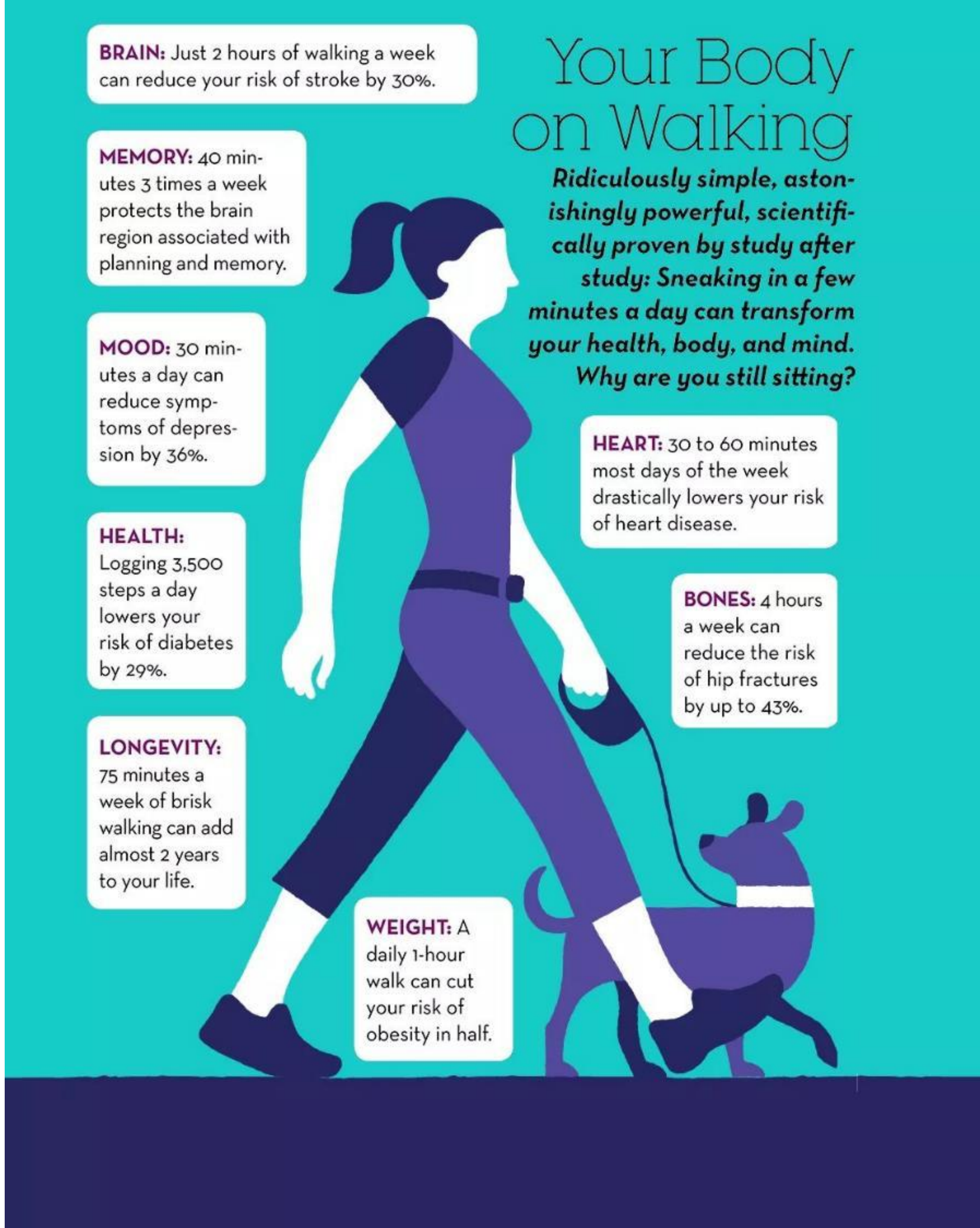
An introduction to NoSQL Databases: document stores

- A different type of database: Mongo DB Introduction
- Using Mongo DB
- Collections of Documents
- Manipulating and Querying Documents

- To see an example of a NoSQL database – a document store that is not queried using SQL.
- To recognise how BSON documents and Python dictionaries can be used.

REMINDER: TAKE A BREAK!

10.30 - 10.40
11.40 – 11.50
12.50 – 13.30
14.30 – 14.40
15.40 – 15.50


Your Body on Walking
Ridiculously simple, astonishingly powerful, scientifically proven by study after study: Sneaking in a few minutes a day can transform your health, body, and mind. Why are you still sitting?
BRAIN: Just 2 hours of walking a week can reduce your risk of stroke by 30%.
MEMORY: 40 minutes 3 times a week protects the brain region associated with planning and memory.
MOOD: 30 minutes a day can reduce symptoms of depression by 36%.
HEALTH: Logging 3,500 steps a day lowers your risk of diabetes by 29%.
LONGEVITY: 75 minutes a week of brisk walking can add almost 2 years to your life.
HEART: 30 to 60 minutes most days of the week drastically lowers your risk of heart disease.
BONES: 4 hours a week can reduce the risk of hip fractures by up to 43%.
WEIGHT: A daily 1-hour walk can cut your risk of obesity in half.