

MANCHESTER
1824

The University of Manchester

Secure Coding – Part 1

COMP16321 – Programming 1

Dr Terence Morley

Department of Computer Science

The University of Manchester



Objectives for this Lecture

- ▶ What is Secure Coding?
- ▶ SQL and the Relational Database, SQLite3
- ▶ Regular Expressions – Regex

What will not be covered

The first things that you might think of when you hear *Secure Coding* are:

- ▶ Passwords
- ▶ Encryption (files and communications)

These are part of Secure Coding, but we will not focus on these aspects here

Personal Anecdote Number One

I once worked with a government department and was given an account to manage certain things and see messages and emails. I bookmarked the website for easy access

After I had stopped working with them and no longer had an account, I clicked on the bookmark and it took me into the system and, strangely, allowed me to see messages and emails for every person that had an account

Personal Anecdote Number One

I contacted them to let them know about the problem but got no reply

I kept checking the website to see if they had fixed it and after two or three days
it no longer let me in

— quietly fixed without acknowledging that a problem ever existed

Personal Anecdote Number One

In the case of my anecdote, security existed in the form of
user accounts with passwords

But something still went wrong

So, there must be more to secure coding than passwords and encryption

What is Secure Coding?

Secure coding is the practice of creating software that doesn't accidentally introduce vulnerabilities

What is Secure Coding?

Definitions of Vulnerability

A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy
[Internet Engineering Task Force]

The existence of a weakness, design, or implementation error that can lead to an unexpected, undesirable event compromising the security of the computer system, network, application, or protocol involved
[European Union Agency for Cybersecurity]

A weakness in design, implementation, operation or internal control
[ISACA]

What is Secure Coding?

Definition of Security

Freedom from risk or danger

We can think of security in the sense of stopping people gaining unauthorised access to systems or data

But, we could also think of it as 'safety'

Personal Anecdote Number Two

I used to create systems for factory automation. Modern factories have very few people in them and the software runs all of the process with little input from humans

In the food industry, they use very large mixers that rotate and/or have blades or rods that perform the mixing. These sometimes need repair or cleaning which is done by a person going inside the mixer

The maintenance workers electrically isolate the mixer, apply a lock and take the key with them. The system can then no longer run the mixer



Personal Anecdote Number Two

However, in one factory, an electrician had inserted a wire into the isolator to bypass it while he was doing some tests and forgot to remove it afterwards

Two men were killed when the system started up
while they were still inside the mixer

In the UK, it used to be that companies would be fined and have to pay
compensation when things like this happen

I believe that the law has changed so that engineers can now be prosecuted
individually

Personal Anecdote Number Two

In this case, it was not the fault of the company I worked with
and was not actually to do with the software

But your software will often be part of a bigger system
and potential dangers should always be kept in mind

Vulnerabilities

What about robustness?

We are not just talking about a simple program that a user runs on their pc

It could be banking software on a web-server or software in a car engine management system

We need to think beyond the thought of just saying that if it crashes the user can re-run it

What is Secure Coding?

Perhaps we could define Secure Coding as:

The practice of designing and developing software so that it does exactly what is supposed to do and doesn't present any risks or dangers, be that to life, property, finance, or reputation, and that doesn't allow for any outside influence to prevent that proper operation

Maybe you can think of a better or more complete definition

Secure Coding?

Some things to keep in mind when developing software or systems:

- ▶ Defensive programming
Think about how things could go wrong
(input validation, attack such as SQL Injection)
- ▶ Open Source development
Others modifying your projects on GitHub, for example
- ▶ Using open source libraries
How do you know they have nothing dodgy in them?

Secure Coding?

Ideas for creating good code

- ▶ Good design practice
- ▶ Good commenting and documentation
- ▶ Use software analysis tools such as lint
- ▶ Code reviews
Have your code checked for problems (in a company)
- ▶ Thorough testing

Secure Coding?

In this lecture, we will

- ▶ Investigate SQL injection attacks
And learn a bit of database programming using Python
- ▶ Learn about using regex for input validation

MANCHESTER
1824

The University of Manchester

Secure Coding – Part 2

COMP16321 – Programming 1

Dr Terence Morley

Department of Computer Science

The University of Manchester



The Relational Database, SQLite

This is a short introduction to the database, SQLite,
and the Structured Query Language (SQL)

This will:

- ▶ Give you an extra tool for your programming toolbox
- ▶ Allow you to understand a couple of security issues

The Relational Database, SQLite



“SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day.”

— <https://www.sqlite.org/>

The Relational Database, SQLite

Pre-compiled binaries for SQLite are available for:

- ▶ Linux
- ▶ Mac OS
- ▶ Windows
- ▶ Android

The source code is available on the website for compilation on other platforms

The Relational Database, SQLite

SQLite is part of the standard Python packages
so you don't need to install it

But, if you want to use the **sqlite3 command shell** (see later)
you may need to install that

Create an SQLite Database File and Table

USERS	
ID	int
NAME	text
AGE	int
ADDRESS	text

NOTE You would normally have the ID as a primary key
but we won't bother with that

Create an SQLite Database File and Table

create_db.py

```
import sqlite3

# Open connection to database. Create the database file if it doesn't exist
conn = sqlite3.connect('test.db')

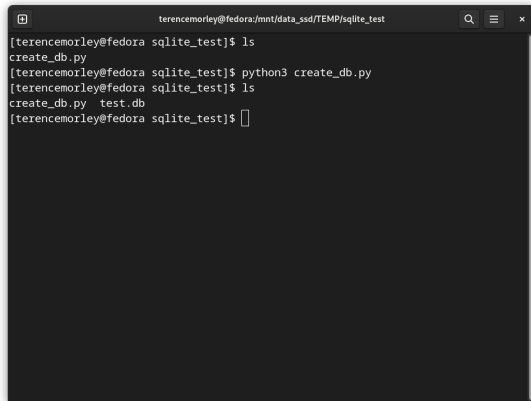
# Delete table 'USERS'. Exception will be thrown if table doesn't exist
try:
    conn.execute('DROP TABLE USERS;')
except:
    pass

# Create a table
conn.execute('CREATE TABLE USERS
             (ID INT,
              NAME TEXT,
              AGE INT,
              ADDRESS TEXT);')

conn.close()
```


Create an SQLite Database File and Table

Running create_db.py

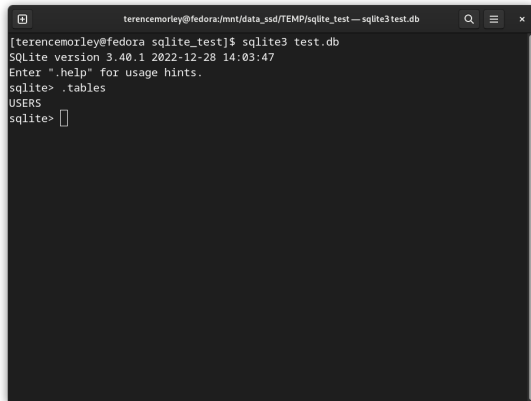


```
terencemorley@fedora:/mnt/data_ssd/TEMP/sqlite_test
[terencemorley@fedora sqlite_test]$ ls
create_db.py
[terencemorley@fedora sqlite_test]$ python3 create_db.py
[terencemorley@fedora sqlite_test]$ ls
create_db.py  test.db
[terencemorley@fedora sqlite_test]$
```

Notice the appearance of the file test.db

Create an SQLite Database File and Table

Looking inside the database file with the sqlite3 command shell

A terminal window titled "terencemorley@fedora: /mnt/data_ssd/TEMP/sqlite_test — sqlite3 test.db". The prompt is "[terencemorley@fedora sqlite_test]\$". The command "sqlite3 test.db" has been entered. The output shows "SQLite version 3.40.1 2022-12-28 14:03:47" and "Enter ".help" for usage hints.". The prompt is now "sqlite>". The command ".tables" has been entered. The output shows "USERS". The prompt is now "sqlite>".

```
terencemorley@fedora: /mnt/data_ssd/TEMP/sqlite_test — sqlite3 test.db
[terencemorley@fedora sqlite_test]$ sqlite3 test.db
SQLite version 3.40.1 2022-12-28 14:03:47
Enter ".help" for usage hints.
sqlite> .tables
USERS
sqlite>
```

It shows the USERS table that was just created

Populate an SQLite Table

populate.py

```
import sqlite3

# Open connection to database. Create the database file if it doesn't exist
conn = sqlite3.connect('test.db')

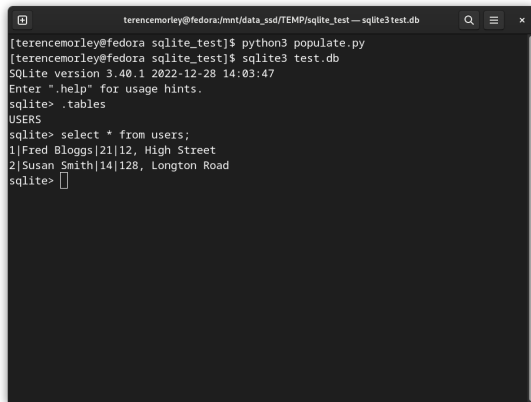
# Populate the table with a couple of records
conn.execute('''INSERT INTO USERS (ID, NAME, AGE, ADDRESS) VALUES
              (1, 'Fred Bloggs', 21, '12, High Street'),
              (2, 'Susan Smith', 14, '128, Longton Road');''')

# Commit the changes
conn.commit()

conn.close()
```

Populate an SQLite Table

Running populate.py and examining the contents of the database file

A terminal window titled 'terencemorley@fedora: /mnt/data_ssd/TEMP/sqlite_test — sqlite3 test.db'. The terminal shows the execution of 'python3 populate.py', opening 'sqlite3 test.db', displaying the SQLite version '3.40.1 2022-12-28 14:03:47', and entering the '.help' command. Then, the '.tables' command is entered, showing 'USERS'. Finally, the 'select * from users;' command is entered, displaying two rows of data: '1|Fred Bloggs|21|12, High Street' and '2|Susan Smith|14|128, Longton Road'.

```
terencemorley@fedora: /mnt/data_ssd/TEMP/sqlite_test — sqlite3 test.db
[terencemorley@fedora sqlite_test]$ python3 populate.py
[terencemorley@fedora sqlite_test]$ sqlite3 test.db
SQLite version 3.40.1 2022-12-28 14:03:47
Enter ".help" for usage hints.
sqlite> .tables
USERS
sqlite> select * from users;
1|Fred Bloggs|21|12, High Street
2|Susan Smith|14|128, Longton Road
sqlite>
```

Notice that SQL command do not start with a dot like the sqlite commands

Query an SQLite Table

query.py

```
import sqlite3

# Open connection to database
conn = sqlite3.connect('test.db')

# Get id number from the user - try inputting 1, 2, %
id = input("Enter a user id: ")

# Create SQL query
qry = "SELECT * FROM USERS WHERE ID LIKE '" + id + "';"
print("\nExecuting:", qry)
print()

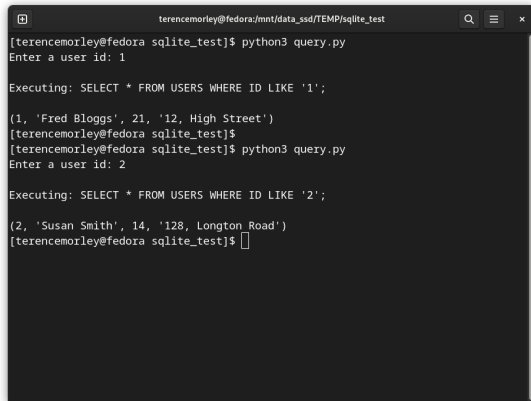
# Query data from the table
rows = conn.execute(qry)

# Display results
for row in rows:
    print(row)

conn.close()
```

Query an SQLite Table

Running query.py and examining the contents of the database file

A terminal window titled 'terencemorley@fedora:/mnt/data_ssd/TEMP/sqlite_test' with search, menu, and close icons. The terminal shows the execution of 'python3 query.py'. It prompts 'Enter a user id: 1', then displays the SQL query 'Executing: SELECT * FROM USERS WHERE ID LIKE '1';' and the result '(1, 'Fred Bloggs', 21, '12, High Street')'. It then prompts 'Enter a user id: 2', displays the SQL query 'Executing: SELECT * FROM USERS WHERE ID LIKE '2';' and the result '(2, 'Susan Smith', 14, '128, Longton Road')'. The prompt returns to '[terencemorley@fedora sqlite_test]\$'.

The program displays the record for the ID specified on the command line

Query an SQLite Table

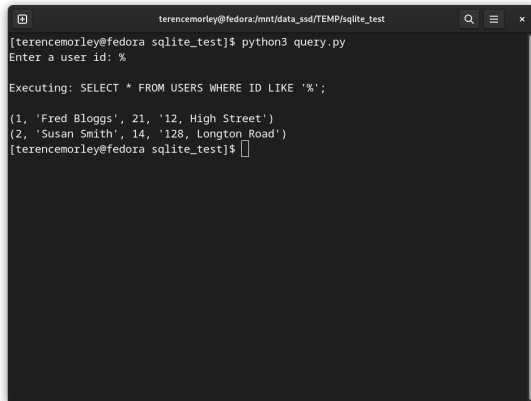
The wildcard character in sqlite is %

A SQL query like this will show **all** records:

```
SELECT * FROM USERS WHERE ID LIKE '%';
```

Query an SQLite Table

A sneaky user somehow gets your program to think that his/her ID is %:



```
terencemorley@fedora:/mnt/data_ssd/TEMP/sqlite_test
[terencemorley@fedora sqlite_test]$ python3 query.py
Enter a user id: %

Executing: SELECT * FROM USERS WHERE ID LIKE '%';

(1, 'Fred Bloggs', 21, '12, High Street')
(2, 'Susan Smith', 14, '128, Longton Road')
[terencemorley@fedora sqlite_test]$
```

Disaster: the sneaky user sees all of your user accounts

Using the SQLite Command Shell

- ▶ Run it (specify the database file name):
`sqlite3 filename.db`
- ▶ Can type commands or SQL queries in the shell
 - ▶ Commands start with a full-stop:
`.tables` (Lists the tables in the database)
 - ▶ SQL queries end with a semi-colon:
`select * from USERS;` (Displays the table named USERS)
- ▶ Quit the shell:
Type in `.quit` or press `ctrl-d`

Security Problem – SQL Injection

Say we implement a query in our code:

```
qry = "SELECT * FROM USERS WHERE ID LIKE '" + id + "';"
```

The program gets a value of **1** from the user and stores it in the variable `id`

Then, the query string, `qry`, contains:

```
SELECT * FROM USERS WHERE ID LIKE '1';
```

Security Problem – SQL Injection

But if user types in:

```
1'; DROP TABLE USERS; --
```

the query will become:

```
SELECT * FROM USERS WHERE ID LIKE '1'; DROP TABLE USERS; --';
```

Security Problem – SQL Injection

The query string then contains these three queries:

```
SELECT * FROM USERS WHERE ID LIKE '1';  
DROP TABLE USERS;  
--';
```

(-- is a comment in SQL)

The hacker guesses that your database has a table called USERS
and causes it to be deleted

Security Problem – SQL Injection

The SQLite `execute()` function only allows one SQL statement to be executed

It has another function for executing multiple queries

Implementing a Game Scoreboard in SQLite

Just for an extra bit of information on using Sqlite in Python and with a slightly more advanced SELECT statement, a basic scoreboard program is shown on the next slide

Implementing a Game Scoreboard in SQLite

game_scoreboard.py

```
import sqlite3

def create_db():
    conn = sqlite3.connect('game.db')
    try:
        conn.execute('CREATE TABLE SCOREBOARD (NAME TEXT, SCORE INT);')
    except:
        pass
    conn.close()

def log_score(name, score):
    conn = sqlite3.connect('game.db')
    conn.execute("INSERT INTO SCOREBOARD (NAME, SCORE) VALUES ('{0}', {1});".format(name, score))
    conn.commit()
    conn.close()

def get_top_scores():
    conn = sqlite3.connect('game.db')
    rows = conn.execute('SELECT * FROM SCOREBOARD ORDER BY SCORE DESC LIMIT 2;')
    print('Top two scores:')
    for row in rows:
        print(row)
    conn.close()

create_db()
log_score('Fred Bloggs', 12800)
log_score('Susan Smith', 28250)
log_score('Abdul Kalam', 14300)
get_top_scores()
```

MANCHESTER
1824

The University of Manchester

Secure Coding – Part 3

COMP16321 – Programming 1

Dr Terence Morley

Department of Computer Science

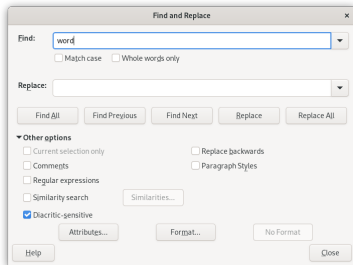
The University of Manchester



Problem: British and American Spelling

Say you want to search a document in a word-processor

You press `ctrl-f` and type in the word you are looking for



But what if the document contains British and American spellings? ...

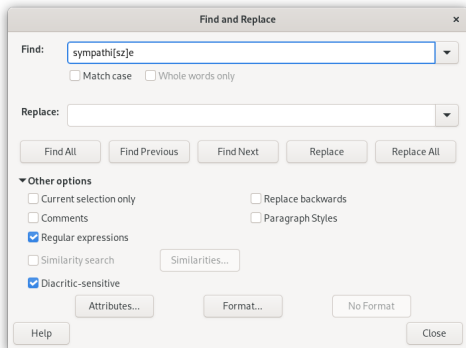
Problem: British and American Spelling

UK	US
sympathise	sympathize
colour	color
dialogue	dialog
...	...

Solution: British and American Spelling

Regular Expressions (Regex)

I sympathise with non-native English speakers because they have to contend with British and American spellings with the latter using the word 'sympathize'. Another example is colour/color.

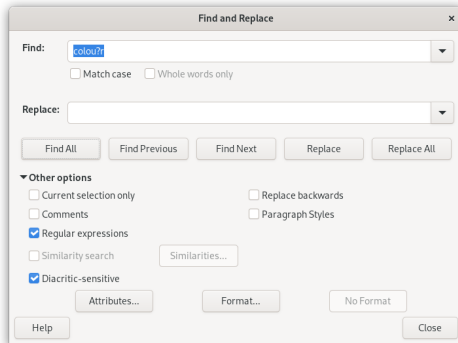


Note: This example uses LibreOffice Writer

Solution: British and American Spelling

Regular Expressions (Regex)

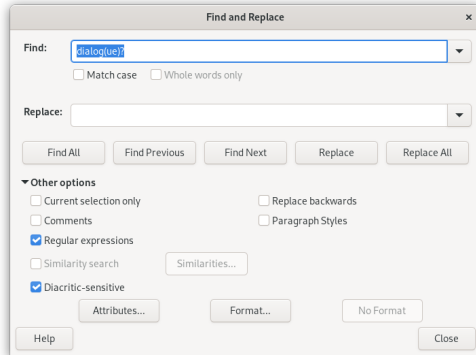
I sympathise with non-native English speakers because they have to contend with British and American spellings with the latter using the word 'sympathize'. Another example is `colour/color`.



Solution: British and American Spelling

Regular Expressions (Regex)

Dialog and dialogue



Solution: British and American Spelling

Regular Expressions (Regex)

UK	US	Regex
sympathise	sympathize	sympathi[sz]e
colour	color	colou?r
dialogue	dialogue	dialog(ue)?

More can be found for regex in LibreOffice Writer's documentation:

<https://help.libreoffice.org/6.2/en-US/text/shared/01/02100001.html>

Regular Expressions (Regex)

What is regex?

A regular expression is a notation that allows for pattern matching in text

It is much more powerful than an ordinary Find function in text editors

It was developed in 1951 by Stephen Cole Kleene
(His surname is pronounced like clay-knee)

It is available in many programming languages, word processors, databases, etc.

Regular Expressions – grep

The Linux command `grep` is used to search for text in files

```
[terencemorley@fedora code]$ grep -E 'colou?r' *  
re_findall.py:text = "The UK write 'colour' and the US write 'color'"  
re_finditer.py:text = "The UK write 'colour' and the US write 'color'"
```

The `-E` option tells it to use regex

The `*` tells it to look in all files in the current directory

regex in Python

The rest of the lecture will give an introduction
to regex with Python

Python Raw Strings

We will be using **backslashes** ('\\') in regex strings

A backslash is normally interpreted as an **escape character**

For example, '\\t' normally gets replaced with a tab character
and '\\n' gives a newline

Prefixing a string with the letter 'r' tells Python that it is a **raw string**:

```
text1 = 'Hello\\t\\t\\tThere'
text2 = r'Hello\\t\\t\\tThere'
print(text1)
print(text2)
```

Python Raw Strings

```
>>> text1 = 'Hello\t\t\tThere'
>>> text2 = r'Hello\t\t\tThere'
>>>
>>> print(text1)
Hello                There
>>>
>>> print(text2)
Hello\t\t\tThere
```

Python re Package

Matching colour/color using re.search()

regex1.py

```
import re

words = ["colour", "color", "colours"]

# Regular expression
regex = r"^colou?r$"

for word in words:
    # Find if the pattern appears in the text
    if(re.search(regex, word)):
        print(word, "-- match")
    else:
        print(word, "-- no match")
```

Output

```
colour -- match
color -- match
colours -- no match
```

Python re Package

The ^ symbol matches with the start of the string

The \$ symbol matches with the end of the string

If the \$ was missed off the end, for example:

```
regex = r"^colou?r"
```

Then the word 'colours' *would* match

Python `re` Package

We just used `re.search()` as returning a bool to check for a match or no match

It actually returns an object which contains information about the match
(see next slide)

We will also look at the the functions `re.findall()` and `re.finditer()`

Python re Package

Function: search

re_search.py

```
import re

text = "In the US they use the word dialog but in the UK, it is spelt dialogue"

# Regular expression
regex = r"dialog(ue)?"

# Find if the pattern appears in the text
if(re.search(regex, text)):
    print("Pattern found")
else:
    print("Pattern not found")

# Find first match in text and its position
match = re.search(regex, text)
print(match.group(0), match.start(), match.end())
```

Output

```
Pattern found
dialog 28 34
```

Python re Package

Function: findall

re_findall.py

```
import re

text = "The UK write 'colour' and the US write 'color'"

# Regular expression
regex = r"colou?r"

# Find all matches in text
matches = re.findall(regex, text)

# Print out each match
for match in matches:
    print(match)
```

Output

```
colour
color
```


Python re Package

Function: finditer

re_finditer.py

```
import re

text = "The UK write 'colour' and the US write 'color'"

# Regular expression
regex = r"colou?r"

# Find all matches in text
matches = re.finditer(regex, text)

for match in matches:
    print(match, "-----", match.group(), "start:", match.start(), "end:", match.end())
```

Output

```
<re.Match object; span=(14, 20), match='colour'> ----- colour start: 14 end: 20
<re.Match object; span=(40, 45), match='color'> ----- color start: 40 end: 45
```

Python re Package

For more information on using the Python library, re, see:

<https://docs.python.org/3/library/re.html>

Python re Package

We have seen how to use the regex package in Python

Next, we will investigate Regular Expressions themselves

MANCHESTER
1824

The University of Manchester

Secure Coding – Part 4

COMP16321 – Programming 1

Dr Terence Morley

Department of Computer Science

The University of Manchester



Regular Expressions in Python

We will now look at various basic uses of regex

Regular Expressions in Python

We saw that '?' matches if the previous character is there or not
as in 'colou?r'

? therefore matches either 0 or 1 occurrences of the preceding character

There are two similar *quantifiers*: + and * as shown on the following slides

Regular Expressions in Python

Matching one or more occurrences with +

regex2.py

```
import re

words = ["tk", "tok", "took"]

# Regular expression
regex = r"^to+k$"

for word in words:
    # Find if the pattern appears in the text
    if(re.search(regex, word)):
        print(word, "-- match")
    else:
        print(word, "-- no match")
```

Output

```
tk -- no match
tok -- match
took -- match
```

Regular Expressions in Python

Matching zero or more occurrences with *

regex3.py

```
import re

words = ["tak", "tk", "tok", "took"]

# Regular expression
regex = r"^to*k$"

for word in words:
    # Find if the pattern appears in the text
    if(re.search(regex, word)):
        print(word, "-- match")
    else:
        print(word, "-- no match")
```

Output

```
tak -- no match
tk -- match
tok -- match
took -- match
```


Regular Expressions in Python

Matching any single character

regex4.py

```
import re

words = ["sit", "sat", "seat", "s6t"]

# Regular expression
regex = r"^s.t$"

for word in words:
    if re.search(regex, word):
        print(word, "-- match")
    else:
        print(word, "-- no match")
```

Output

```
sit -- match
sat -- match
seat -- no match
s6t -- match
```

Regular Expressions in Python

The `.` in the previous slide matches any character (including numbers)

If you want to match only certain letters (a, e, i), you use `[aei]`

This means match a character if it is a, e, or i

Regular Expressions in Python

Matching any single character from a selection

regex5.py

```
import re

words = ["sit", "sat", "seat", "sot"]

# Regular expression
regex = r"^s[aei]t$"

for word in words:
    if re.search(regex, word):
        print(word, "-- match")
    else:
        print(word, "-- no match")
```

Output

```
sit -- match
sat -- match
seat -- no match
sot -- no match
```

Regular Expressions in Python

You can also use the `[]` notation with ranges

For example, to match any letter, uppercase or lowercase,
you can use `[A-Za-z]`

Regular Expressions in Python

Matching numerical digits with `\d`

regex6.py

```
import re

words = ["a1", "b5", "v62"]

# Regular expression
regex = r"^[A-Za-z]\d$"

for word in words:
    if re.search(regex, word):
        print(word, "-- match")
    else:
        print(word, "-- no match")
```

Output

```
a1 -- match
b5 -- match
v62 -- no match
```

Notice that `\d` matches single numerical digits, not general numbers like 62

Regular Expressions in Python

So, how do we find multiple numerical digits?

We could look for one or more digits with `\d+`

But we will investigate another quantifier `{n}`

which matches exactly `n` occurrences of the preceding character

Regular Expressions in Python

Matching exactly n occurrences with {n}

regex7.py

```
import re

words = ["a5", "b15", "v300"]

# Regular expression
regex = r"^[A-Za-z]\d{3}$"

for word in words:
    if re.search(regex, word):
        print(word, "-- match")
    else:
        print(word, "-- no match")
```

Output

```
a5 -- no match
b15 -- no match
v300 -- match
```

Regular Expressions in Python

As well as using the quantifier `{n}`

to match exactly `n` occurrences of the preceding character,

we can use `{m,n}` to match a number of occurrences between `m` and `n`,
inclusively

Regular Expressions in Python

Matching between m and n occurrences with {m,n}

regex8.py

```
import re

words = ["a5", "b15", "v3000", "z12345"]

# Regular expression
regex = r"^[A-Za-z]\d{2,4}$"

for word in words:
    if re.search(regex, word):
        print(word, "-- match")
    else:
        print(word, "-- no match")
```

Output

```
a5 -- no match
b15 -- match
v3000 -- match
z12345 -- no match
```

Regular Expressions in Python

Matching options with | (meaning 'or')

regex9.py

```
import re

text = "I have got a mouse, a dog and a cat. Oh, I forgot another cat."

# Regular expression
regex = r"dog|cat"

# Find all matches in text
matches = re.findall(regex, text)

# Print out the matches
print(matches)
```

Output

```
['dog', 'cat', 'cat']
```

Regular Expressions in Python

Matching full words in a string with the word boundary `\b`

regex10.py

```
import re

text = "I call my dog, doggy."

# Regular expression
regex = r"\bdog\b"

# Find all matches in text
matches = re.findall(regex, text)

# Print out the matches
print(matches)
```

Output

```
['dog']
```

Doesn't match 'doggy' because there needs to be a word boundary after 'dog'

Regular Expressions in Python

Excluding options with `[^...]`

Match CS units except first year ones

regex11.py

```
import re

text = "My favourite units were COMP16321, COMP27112, COMP37212"

# Regular expression
regex = r"\bCOMP[^1]\d{4}\b"

# Find all matches in text
matches = re.findall(regex, text)

# Print out the matches
print(matches)
```

Output

```
['COMP27112', 'COMP37212']
```

Regular Expressions in Python

Validating Computer Science unit codes

Unit codes begin with COMP and are followed by 5 digits

The first digit is the year: 1, 2, or 3

The last digit is the semester: 1, 2, or 0 (for all year)

Regular Expressions in Python

Validating Computer Science unit codes

regex12.py

```
import re

text = "COMP16321, COMP56321, COMP27112, COMP37212, COMP3721, COM27112"

# Regular expression
regex = r"\bCOMP[1-3]\d{3}[0-2]\b"

# Find all matches in text
matches = re.findall(regex, text)

# Print out the matches
print(matches)
```

Output

```
['COMP16321', 'COMP27112', 'COMP37212']
```

Regular Expressions in Python

Replacing text using re.sub()

regex13.py

```
import re

text = "David has got a dog, Ahmed has got a cat, and Frida owns a mouse."

# Regular expression
regex = r"\bcat|dog|mouse\b"

# Replace all matches with 'pet'
new_text = re.sub(regex, 'pet', text)

print(text)
print(new_text)
```

Output

```
David has got a dog, Ahmed has got a cat, and Frida owns a mouse.
David has got a pet, Ahmed has got a pet, and Frida owns a pet.
```

100

Validating email addresses

A simple solution is as follows:

```
r"([a-zA-Z0-9_+.-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+"$)
```

But, this probably won't get all addresses

This is supposed to match 99.99% of email addressses:

```
(?:[a-z0-9!#$%&'*/+=?^_`{|}~]+(?:\. [a-z0-9!#$%&'*/+=?^_`{|}~]+)*"?(?:\[x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x2b\x2d-\x2f\]|\[\[x01-\x09\x0b\x0c\x0e-\x2f\])*"?)@(?: (?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?)\. )+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?)?[\[(?: (?:25[0-5]|2[0-4][0-9]|01[0-9]?[0-9][0-9]?)\.) {3}(?:25[0-5]|2[0-4][0-9]|01[0-9]?[0-9][0-9]?)?[a-z0-9-]*[a-z0-9](?:[x01-\x08\x0b\x0c\x0e-\x1f\x21-\x2b\x2d-\x2f\]|\[\[x01-\x09\x0b\x0c\x0e-\x2f\])+\)]\]
```

— RFC 5322 Official Standard

Regular Expressions in Python

Validating email addresses

regex14.py

```
import re

words = ["fred.boggs123@somewhere.co.uk", "here.com"]

# Regular expression
regex = r'''(?:[a-z0-9!#$%&'*/+=?^_`{|}~-\.]+(?:\.[a-z0-9!#$%&'*/+=?^_`{|}~-\.]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\\])'''

for word in words:
    # Find if the pattern appears in the text
    if re.search(regex, word):
        print(word, "-- match")
    else:
        print(word, "-- no match")
```

Output

```
fred.boggs123@somewhere.co.uk -- match
here.com -- no match
```

regex Metacharacters

Character	Description	Example
[]	A set of characters	"[aeiou]" or "[a-z]"
\	Signals a special sequence	"\d"
.	Any character (except newline)	"s.t"
^	Start of string	"^hello"
\$	End of string	"world\$"
*	Zero or more occurrences	"abc*"
+	One or more occurrences	"abc+"
?	Zero or one occurrence	"dogs?"
{ }	Number of occurrences	"\d{5}" or "\d{2,5}"
	Or	"staff student"
()	Capture and Group	"dialog(ue)"

regex Character Classes

Character	Description
\A	Matches the beginning of a string (but not an internal line)
\b	Matches word boundaries
\B	Matches characters which are not on word boundaries
\D	Matches non-digits
\s	Matches whitespace
\S	Matches non-whitespace characters
\w	Matches alphanumeric, including '_'. Equivalent to [A-Za-z0-9_]
\W	Matches non-alphanumeric characters, excluding '_'
\Z	Matches the end of a string (but not an internal line)

Summary

What have we discussed?

- ▶ Secure Coding
- ▶ SQL and the Relational Database, SQLite3
- ▶ Regular Expressions – Regex