ANSWERS

A12.9 Web technologies and search

Networks and the web · A level · OCR H446 1.3.4 · about 30 min

What this lesson is about

HTML, CSS and JavaScript, client and server side processing, indexing and PageRank.

- HTML
- CSS
- JavaScript
- Client side and server side
- Search engine indexing
- PageRank

Questions

6 marks in all

1. Which CSS selector styles every element that has class="note"? [1 mark]

- ☐ **A** .note
- ☐ **B** #note
- ☐ **C** note
- ☐ **D** <note>

> **Answer:** A. A dot selects a class; a hash selects the single element with that id.

2. Which HTML tag makes a hyperlink? [1 mark]

- ☐ **A** <a href="...">
- ☐ **B** <link>
- ☐ **C** <p>
- ☐ **D**

> **Answer:** A. The a tag with an href attribute links to another page; link in the head connects a stylesheet.

3. In the PageRank formula with d = 0.85, page A is linked to only by page B. B has a rank of 2 and 4 outbound links. What is PR(A)? Give it to 3 decimal places. [1 mark]

> **Answer:** 0.575 (accept within 0.0005). PR(A) = 0.15 + 0.85 x (2 / 4) = 0.15 + 0.425 = 0.575.

4. Which are reasons to process a form on the server rather than only in the browser? Choose all that apply. [1 mark]

Tick every answer that is true.

- ☐ **A** Client-side code can be seen, changed or switched off by the user
- ☐ **B** The server can check the data against its database
- ☐ **C** It gives instant feedback with no request
- ☐ **D** It reduces the load on the server

> **Answer:** A, B. Server-side checks cannot be bypassed and can use the database; instant feedback and lower server load are client-side advantages.

5. What does this program print? [1 mark]

```
index = {}
pages = {'p1': 'robot kit', 'p2': 'robot sim', 'p3': 'kit list'}
for page, words in pages.items():
    for word in words.split():
        index.setdefault(word, []).append(page)
print(index['kit'])
```

Answer:

```
['p1', 'p3']
```

The index maps each word to the pages containing it; kit appears in p1 and p3.

6. What does a search engine's crawler do? [1 mark]

- ☐ **A** Visits pages and follows their links to find pages to add to the index
- ☐ **B** Ranks pages by how many words they contain
- ☐ **C** Blocks pages that contain malware
- ☐ **D** Runs JavaScript on the server

Answer: A. Crawlers (spiders) collect pages ahead of time so searches can use the index instead of the live web.

The task: index and rank

A small website has five pages. `pages` maps each page name to its text; `links` maps each page name to the list of pages it links to. Every page links to at least one other. 1. Build an index: a dictionary mapping each word (split the text on spaces) to the list of pages that contain it, with no page listed twice for one word. 2. Work out the PageRank of every page with d = 0.85. Start every page at 1.0 and do exactly 30 iterations, calculating each iteration's ranks only from the previous iteration's ranks. 3. For each page, in the order of `pages`, print `<page> <rank>` with the rank to 2 decimal places. 4. For each of the words `robot` and `simulator`, in that order, print `search <word>: <pages>`, the pages containing that word from highest rank to lowest, separated by single spaces. That is 7 lines. The robot does not drive.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

pages = {
    "home": "bugbot lessons for your robot in python",
    "sim": "the robot simulator runs python in the browser",
    "kit": "buy a robot kit for your school",
    "blog": "news about the simulator and the kit",
    "faq": "questions about python and the robot",
}
links = {
    "home": ["sim", "kit", "faq"],
    "sim": ["home"],
    "kit": ["home", "sim"],
    "blog": ["sim", "kit"],
    "faq": ["home", "sim"],
}
D = 0.85
rank = {page: 1.0 for page in pages}
```

The hint students can ask for: Work every new rank from the old ranks, not from ranks you have already updated this round: build a fresh dictionary each iteration. A page's share of its vote is its rank divided by how many links leave it. For the search, find the pages whose word lists hold the query, then sort them by rank, highest first.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

pages = {
    "home": "bugbot lessons for your robot in python",
    "sim": "the robot simulator runs python in the browser",
    "kit": "buy a robot kit for your school",
    "blog": "news about the simulator and the kit",
    "faq": "questions about python and the robot",
}
links = {
    "home": ["sim", "kit", "faq"],
    "sim": ["home"],
    "kit": ["home", "sim"],
    "blog": ["sim", "kit"],
    "faq": ["home", "sim"],
}
```

```
D = 0.85

index = {}
for page, text in pages.items():
    for word in text.split():
        if word not in index:
            index[word] = []
        if page not in index[word]:
            index[word].append(page)

rank = {page: 1.0 for page in pages}
for iteration in range(30):
    new = {}
    for page in pages:
        total = 0.0
        for other in pages:
            if page in links[other]:
                total = total + rank[other] / len(links[other])
        new[page] = (1 - D) + D * total
    rank = new

for page in pages:
    print(f"{page} {rank[page]:.2f}")
for word in ["robot", "simulator"]:
    found = sorted(index.get(word, []), key=lambda p: rank[p], reverse=True)
    print(f"search {word}: {' '.join(found)}")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.