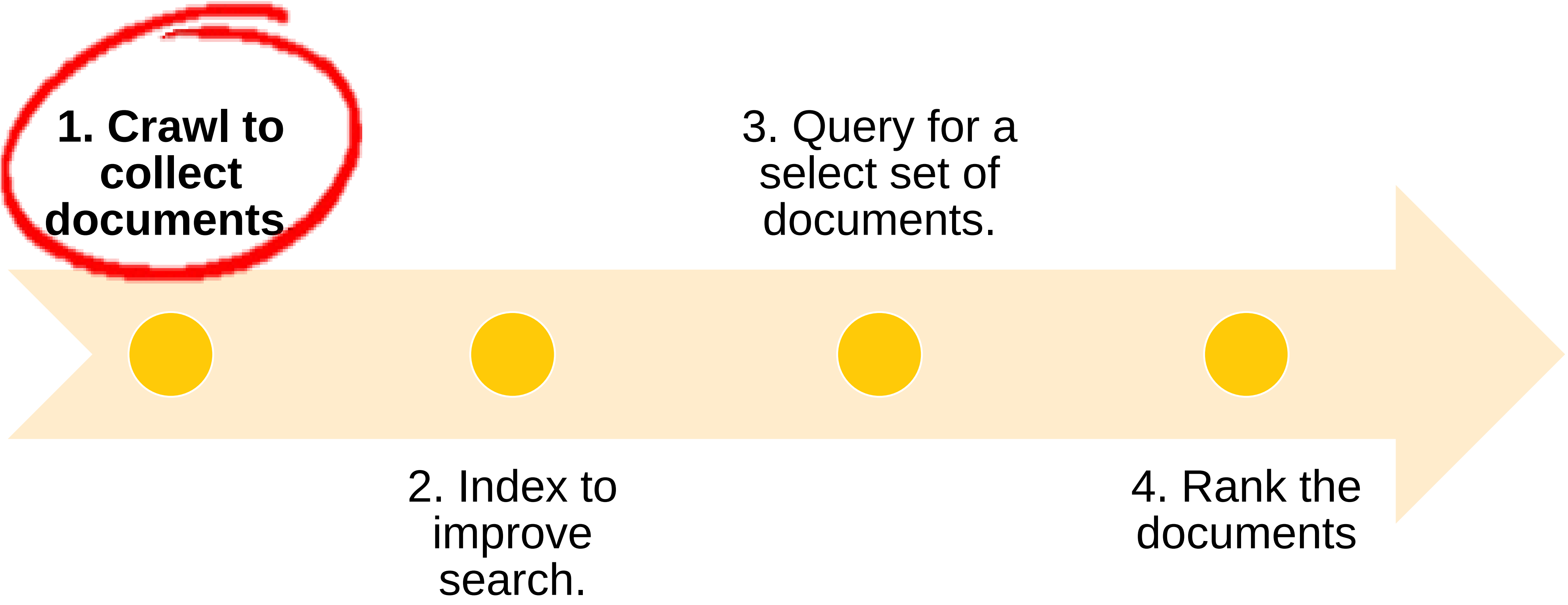
ENGR 102 PROGRAMMING PRACTICE

WEEK 11

İSTANBUL ŞEHİR UNIVERSITY

Searching & Ranking

Search Engine


1. Crawl to collect documents.
2. Index to improve search.
3. Query for a select set of documents.
4. Rank the documents

Search Engine

- Create a Python module (`mysearchengine.py`).

- The module will have two classes:
  - one for crawling and creating the database, and
  - the other for doing full-text searches by querying the database, as well as ranking.

Crawler Class

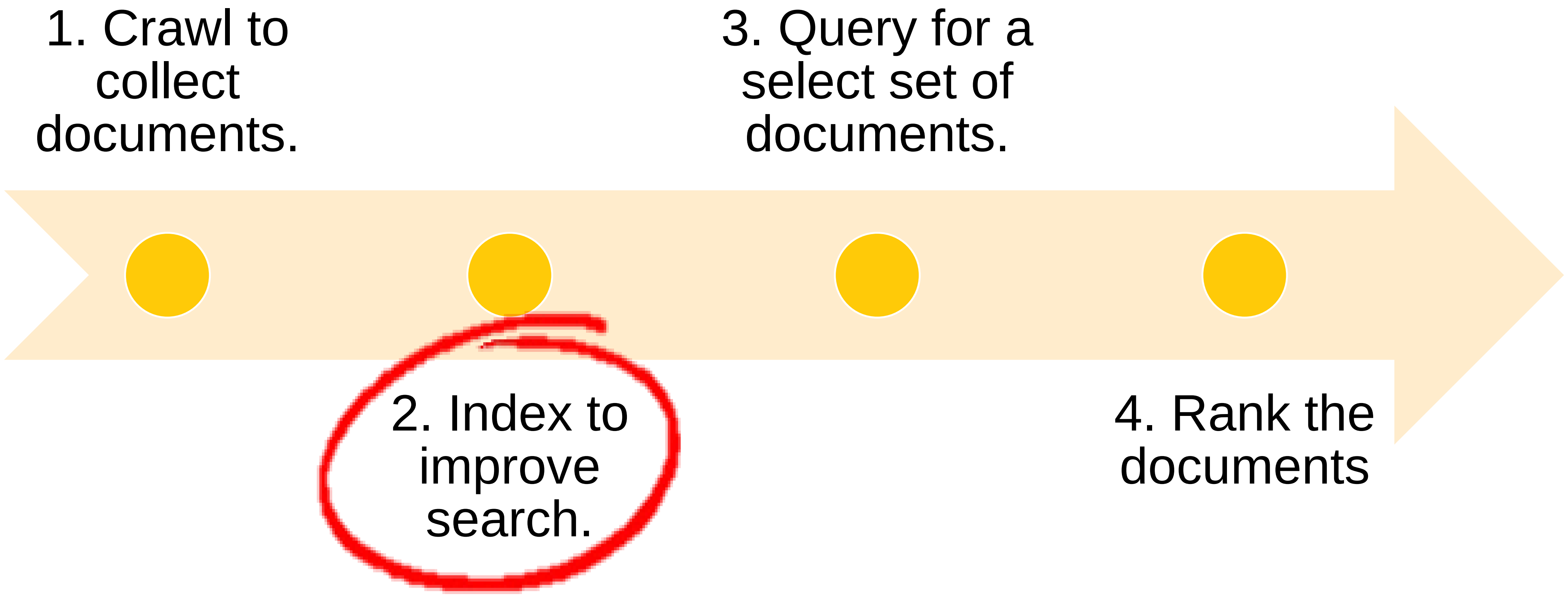
```
class Crawler:
  # Initialize the crawler with the names of database tables
  def __init__(self, dbtables):
    pass

  # Starting with a list of pages, do a breadth-first search
  # to the given depth, indexing pages as we go
  def crawl(self, pages, depth=2):
    pass

  # Index an individual page
  def addtoindex(self, url, soup):
    pass

  # Extract the text from an HTML page (no tags)
  def gettextonly(self, soup):
    pass

  # Separate the words by any non-alphanumeric character
  def separatewords(self, text):
    pass
```

Crawling pages - crawl( )

```
def crawl(self, pages, depth=2):
    for i in range(depth):
        newpages = set()
        for page in pages:
            r = requests.get(page)
            soup = BeautifulSoup(r.content)
            if not self.addtoindex(page, soup):
                continue

            links = soup.find_all('a')
            for link in links:
                if 'href' in link.attrs:
                    url = urljoin(page, link['href'])

                    if not self.isindexed(url):
                        newpages.add(url)

                        linkText = self.gettextonly(link)
                        self.addlinkref(page, url, linkText)

        pages = newpages
```

End loop

Search Engine


1. Crawl to collect documents.
2. Index to improve search.
3. Query for a select set of documents.
4. Rank the documents

Setting Up Database

<u>Four dictionaries:</u>

- **urllist** is the list of URLs that have been indexed.
  {url: outgoing_link_count}

- **wordlocation** is a list of the locations of words in the documents.
  {word: {url: [loc1, loc2, ..., locN]}}

- **link** stores two URL IDs, indicating a link from one page to another.
  {tourl: {fromUrl: None}}

- **linkwords** store words that are included in a link.
  {word: [(urlFrom1, urlTo1), ..., (urlFromN, urlToN)]}

Building the Database

- The database will be stored using shelve module
- Provides persistent object storage on disk
- Similar to dbm, but more practical
- Use with import shelve

shelve – Persistent storage of arbitrary Python objects

- Key-value structure (like a dictionary)
- Persists data on disk (like dbm)
- Keys may only be strings (like dbm)
- Values may be any object (unlike dbm, like a dictionary)
    - No need to pickle objects
- Handles updates automagically

shelve – open and insert data

```
import shelve


s = shelve.open('test_shelf.db')

s['key1'] = {'int': 10, 'float':9.5, 'string':'data'}

s.close()
```

shelve – read existing content

```
import shelve


s = shelve.open('test_shelf.db')
existing = s['key1']
print(existing)
s.close()
# prints: {'int': 10, 'float': 9.5, 'string': 'data'}
```

shelve – auto update with writeback = True

```
import shelve
s = shelve.open('test_shelf.db')
print(s['key1'])
s['key1']['new_value'] = 'this was not here before'
s.close()

s = shelve.open('test_shelf.db')
print(s['key1'])
s.close()
# prints: {'int': 10, 'float': 9.5, 'string': 'data'}
          {'int': 10, 'float': 9.5, 'string': 'data'}
```

shelve – auto update with writeback = True

```
import shelve
s = shelve.open('test_shelf.db', writeback = True)
print(s['key1'])
s['key1']['new_value'] = 'this was not here before'
s.close()
# prints: {'int': 10, 'float': 9.5, 'string': 'data'}

s = shelve.open('test_shelf.db', writeback = True)
print(s['key1'])
s.close()
# prints: {'int': 10, 'new_value': 'this was not here
            before', 'float': 9.5, 'string': 'data'}
```

Setting Up the Database

```
import mysearchengine


pagelist=['http://sehir.edu.tr/en']

dbtables = {'urllist': 'urllist.db',
            'wordlocation':'wordlocation.db',
            'link':'link.db', 'linkwords':'linkwords.db',
            'pagerank':'pagerank.db'}

crawler = mysearchengine.Crawler(dbtables)
crawler.createindextables()
```

createindextables( )

```
# Create the database tables
def createindextables(self):
    # {url:outgoing_link_count}
    self.urllist = shelve.open(self.dbtables['urllist'], writeback=True, flag='c')

    #{word:{url:[loc1, loc2, ..., locN]}}
    self.wordlocation = shelve.open(self.dbtables['wordlocation'], writeback=True, flag='c')

    #{tourl:{fromUrl:None}}
    self.link = shelve.open(self.dbtables['link'], writeback=True, flag='c')

    #{word:[(urlFrom, urlTo), (urlFrom, urlTo), ..., (urlFrom, urlTo)]}
    self.linkwords = shelve.open(self.dbtables['linkwords'], writeback=True, flag='c')
```

Inserting into the Database

Get a list of words on the page. → Add the page and all the words to the index. → Create links between them with their locations in the document.

Finding Words on a Page

- The files on the Web are HTML and thus contain a lot of tags, properties, etc.
- The first step is to extract all the parts of the page that are text.
- You can do this by searching the soup for text nodes and collecting all their content.

Crawling pages - crawl( )

```
def crawl(self, pages, depth=2):
    for i in range(depth):
        newpages = set()
        for page in pages:
            r = requests.get(page)
            soup = BeautifulSoup(r.content)
            if not self.addtoindex(page, soup):
                continue

            links = soup.find_all('a')
            for link in links:
                if 'href' in link.attrs:
                    url = urljoin(page, link['href'])

                    if not self.isindexed(url):
                        newpages.add(url)

                    linkText = self.gettextonly(link)
                    self.addlinkref(page, url, linkText)

        pages = newpages
```

End loop

Extract the text on a page

```
# Extract the text from an HTML page (no tags)
def gettextonly(self, soup):
    v = soup.string
    if v == None:
        c = soup.contents
        resulttext = ''
        for t in c:
            subtext = self.gettextonly(t)
            resulttext += subtext + '\n'
        return resulttext
    else:
        return v.strip()
```

Finding the Words

- Split a string into a list of separate words so that they can be added to the index.
- Our approach:
    - Consider anything that isn't a letter or a number to be a separator.
- You can do this using a regular expression.

Separating into Words

```
# Separate the words by any non-alphanumeric character
def separatewords(self, text):
    splitter = re.compile('\\W+')
    return [s.lower() for s in splitter.split(text) if s != '']
```

[^a-zA-Z0-9_]

Crawling pages - crawl( )

```
def crawl(self, pages, depth=2):
    for i in range(depth):
        newpages = set()
        for page in pages:
            r = requests.get(page)
            soup = BeautifulSoup(r.content)
            if not self.addtoindex(page, soup):
                continue

            links = soup.find_all('a')
            for link in links:
                if 'href' in link.attrs:
                    url = urljoin(page, link['href'])

                    if not self.isindexed(url):
                        newpages.add(url)

                        linkText = self.gettextonly(link)
                        self.addlinkref(page, url, linkText)

        pages = newpages
```

End loop

Checking if this page is already indexed

```
# Return true if this url is already indexed
def isindexed(self, url):
    cleaned_url = smart_str(url)
    # urllist = {url:outgoing_link_count}
    if cleaned_url in self.urllist:
        return True
    else:
        return False
```

Crawling pages - crawl( )

```
def crawl(self, pages, depth=2):
    for i in range(depth):
        newpages = set()
        for page in pages:
            r = requests.get(page)
            soup = BeautifulSoup(r.content)
            if not self.addtoindex(page, soup):
                continue

            links = soup.find_all('a')
            for link in links:
                if 'href' in link.attrs:
                    url = urljoin(page, link['href'])

                    if not self.isindexed(url):
                        newpages.add(url)

                        linkText = self.gettextonly(link)
                        self.addlinkref(page, url, linkText)

        pages = newpages
```

End loop

Adding into the index

```
# Index an individual page
def addtoindex(self, url, soup):
    if self.isindexed(url):
        print('skip', url, 'already indexed')
        return False

    print('Indexing ' + url)

    # Get the individual words
    text = self.gettextonly(soup)
    words = self.separatewords(text)

    # Record each word found on this page
    for i in range(len(words)):
        word = smart_str(words[i])

        if word in ignorewords:
            continue

        #{word:{url:[loc1, loc2, ..., locN]}}
        self.wordlocation.setdefault(word, {})
        self.wordlocation[word].setdefault(url, [])
        self.wordlocation[word][url].append(i)
    return True
```