COMP10001 Foundations of Computing
Semester 1, 2019

Tutorial Questions: Week 11

— VERSION: 1474, DATE: MAY 16, 2019 —

Discussion

1. Revise "recursion". What is it and what makes it useful? How is a recursive function written?

   A: *Recursion is where a function calls itself repeatedly to solve a problem. Rather than using a loop to process something, a recursive function usually calls itself with a smaller or broken-down version of the input until it reaches the answer.*
   Recursion can be useful in implementing certain algorithms, and in solving problems where the amount of nesting required is proportional to the size of the input.
   A recursive function is written the same as any other function except it contains a function call to itself.

2. What are the differences and similarities between recursion and iteration?

   A: *Recursion and iteration can both repeat actions multiple times over a collection of values. Iteration works by moving from one item to the next until the end is reached. Recursion works by repeatedly calling the same function with less input each time until there is none left.*
   You may think of the difference between the two being that while iteration repeats actions over or across a collection of values, recursion iterates "into" the collection, in a series of function calls with a depth proportional to the size of the collection. There are visualisations which would help with this.

3. Recursive functions can use multiple recursive calls. How would a function with a single recursive call execute differently compared to one with multiple calls?

   A: *Both functions would give the same answer and calculate it in a similar way, the difference is in the structure of calls that results. A recursive function with a single recursive call would have a call structure resembling a "stick". That is, a single series of function calls, each one triggering the next. With multiple calls, the structure resembles a "tree", where the input is split in half each time, branching out into two calls. This structure is less deep than a stick and may look more complicated than a stick, but is actually much more efficient when it comes to recursively storing and retrieving information (beyond the scope of this subject but you'll learn more in COMP10002).*

   Now try Exercise 1

4. What is a "URL"? What are the components of one?

   A: *A URL (Uniform Resource Locator) is a way of addressing resources online. It identifies the "path" to a particular file or resource, accessed via a particular "scheme", located at a "host" communicated to via a "port".*
   A URL is a string containing this information: scheme://hostname:port/path

5. What is "HTML"? What is is used for?

   A: *HTML (Hypertext Markup Language) is a markup language (not a programming language like Python) used to take a document composed of text and other media and communicate how it is to be displayed and rendered. It is used extensively on the web: an internet user displays a webpage by downloading the website's HTML which is then rendered and displayed to the user by the browser. Hyperlinks and other web content can be stored in HTML.*

6. Revise the HTML tags you've learned about. How do we use them to format our document?

   A: *Tags are <angle bracket delimited>commands which give instruction about how a document is to be formatted. Often there will be an "opening" <tag>and "closing" </tag>pair of tags where the second includes a slash to show it is closing the first. This communicates that text between the tags should be displayed or interpreted in a particular way. Tags which we've covered include text formatting <b>(bold text), <i>(italic text) and <u>(underline); structural tags such as <html>(covers an entire html document), <head>(header section) and <body>(the body content of a document); lists and tables such as <ul>(unordered bullet-point list), <ol>(ordered list), <li>(delimiting a single list item), <table>(holding a table) which contains <tr>(table rows) and <td>(table cells); and media tags <a>(links to a URL), <img>(an image), <audio>(some audio) and <video>(a video).*

7. What is an HTML "entity"? Why are they needed?

   A: *An entity is a special character which must be written with a special `&entity;` syntax. They include the angle brackets which form part of HTML syntax and would therefore be impossible to include as simple characters without the use of entities (< is `&lt;` and > is `&gt;`).*

8. What is the difference between a static and dynamic HTML page?

   A: *A static page is one which does not change each time it is sent to a user who requests it. This is in contrast to a dynamic HTML page, in which the contents are generated by a computer (allowing them to be tailored to the user requesting it) before sending. An example of a static HTML page is an information page for a product which has no need to be personalised as it should be the same for any customer who looks at it. A dynamic HTML page could be Grok, since your name is displayed in the top corner which will be different for every person using it: Grok's system inserts your name there before it sends you the HTML document.*

   Now try Exercises 2-3

Exercises

1. Convert the following iterative function into a recursive function:

```
def base_change(n, base):
    retval = 0
    n = str(n)[::-1]
    for i in range(len(n)):
        retval += base**i * int(n[i])
    return retval
```

A:

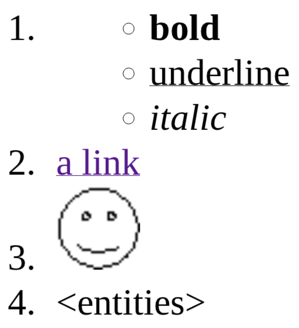
```
def base_change_rec(n, base):
    if len(n) == 0:
        return 0
    return int(n[0])*base**(len(n)-1) + base_change_rec(n[1:],base)

def base_change(n, base):
    return base_change_rec(str(n),base)
```

2. Fill in the blanks of the HTML code on the following page so that it produces the output on the right when opened in a browser. You may use each of the options provided more than once.

3. **(Technology question)** Try visiting your favourite website and viewing the HTML source. How many tags can you recognise? Try editing the source to change the appearance of the webpage (this change will only occur locally on your computer). With knowledge of HTML, you can do a lot more on the web!

```
<               html>
<      >
  <      >
    <       >The Title</      >
  </      >
  <      >
    <      >
      <      >
        <      >
          <      ><      >bold</      ></      >
          <      ><      >underline</      ></      >
          <      ><      >italic</      ></      >
        </      >
      </      >
      <      ><         =''>a link</      ></      >
      <      >
      <         ='smiley.gif' alt='smiley'/>
      </      >
      <      >    entities     </      >
    </      >
  </      >
</      >
```



1. ○ **bold**
   ○ underline
   ○ *italic*
2. a link
3.
4. <entities>

i	li	&gt;	b	head
ol	img src	ul	a href	u
html	&lt;	title	!DOCTYPE	body

A:

```
<!DOCTYPE html>
<html>
  <head>
    <title>The Title</title>
  </head>
  <body>
    <ol>
      <li>
        <ul>
          <li><b>bold</b></li>
          <li><u>underline</u></li>
          <li><i>italic</i></li>
        </ul>
      </li>
      <li><a href=''>a link</a></li>
      <li><img src='smiley.gif'/></li>
      <li>&lt;entities&gt;</li>
    </ol>
  </body>
</html>
```

Problems

1. Write a recursive function which takes an integer `n` and calculates the n^{th} fibonacci number. The 0^{th} fibonacci number is 0, the 1^{st} fibonacci number is 1 and all following fibonacci numbers are defined as the sum of the preceding two fibonacci numbers.

   A:

   ```
   def fibonacci(n):
       if n == 0:
           return 0
       elif n == 1:
           return 1
       else:
           return fibonacci(n-1) + fibonacci(n-2)
   ```

2. Write a function which takes a URL as a string and reads it, splitting it into its components (scheme, hostname...) and returning them in a tuple.

A:

```
def split_URL(url):
    scheme = hostname = port = path = ''
    scheme_i = host_i = port_i = 0
    i = 0
    while i < len(url):
        if url[i:i+3] == '://':
            scheme = url[:i]
            scheme_i = i = i + 3 # skip past ://
            break
        i += 1
    while i < len(url):
        if url[i] == ':':
            hostname = url[scheme_i:i]
            host_i = i = i + 1 # skip past :
            break
        i += 1
    while i < len(url):
        if url[i] == '/':
            port = url[host_i:i]
            port_i = i + 1 # skip past /
            break
        i += 1
    path = url[port_i:]
    return (scheme, hostname, port, path)
```

3. Write a function which takes a list of lists containing cell values and optionally two lists specifying row and column headings, and formats the cell values into a HTML table, returning the HTML text as a string.

A:

```
def array2table(array, row_headings=None, col_headings=None):
    def generate_cell(value, tag="td"):
        return f"<{tag}>{value}</{tag}>"

    def generate_row(row, tag="td", first_cell=""):
        row_out = f"<tr>{first_cell}"
        for element in row:
            row_out += generate_cell(element, tag)
        return row_out + "</tr>"

    out = "<table>"
    if col_headings:
        first_cell = ''
        if row_headings:
            assert len(row_headings) == len(array)
            first_cell = generate_cell("&nbsp;")
        out += generate_row(col_headings, tag="th", first_cell=first_cell)
    for i in range(len(array)):
        first_cell = ''
        if row_headings:
            first_cell = generate_cell(row_headings[i], tag="th")
        out += generate_row(array[i], first_cell=first_cell)
    return out + "</table>"
```