In the following we focus on client requests that are generated using HTML forms

**Request methods: GET versus POST**

The two main request methods used with HTML forms are **GET** and **POST**:

- **GET**:
  - Form data is appended to the URI in the request (limited to 1KB to 8KB characters depending on both browser and server)
  - Form data is accessed by the CGI program via environment variables, name/value pairs that are part of the environment in which a process/programs is run by the operating system
  - Requests remain in the browser history and can be bookmarked
  - Requests should not be used for sensitive data, e.g. passwords

- **POST**:
  - Form data is appended to end of the request (after headers and blank line)
  - There is no limit on the length/size of the form data
  - Form data can be accessed by the CGI program via standard input
  - Form data is not necessarily URL-encoded (but URL-encoding is the default)
  - Requests do not remain in the browser history and cannot be bookmarked
  - Requests are suitable for the transfer of sensitive data, e.g. passwords

---

**Client requests**

In the following we focus on client requests that are generated using HTML forms

```html
<html lang="en-GB">
<head><title>My HTML Form</title></head>
<body>
<form action="http://student.csc.liv.ac.uk/cgi-bin/cgiwrap/uh/process" method="post">
  <label>Enter your user name: <input type="text" name="username"></label>
  <br>
  <label>Enter your full name: <input type="text" name="fullname"></label>
  <br>
  <input type="submit" value="Click for response">
</form>
</body>
</html>
```
Environment variables: GET

**Env variable** | **Meaning**
---|---
QUERY_STRING | The query information passed to the program
REQUEST_METHOD | The request method that was used
PATH_INFO | Extra path information passed to a CGI program
PATH_TRANSLATED | Translation of PATH_INFO from virtual to physical path
SCRIPT_NAME | The relative virtual path of the CGI program
SCRIPT_FILENAME | The physical path of the CGI program

```python
GET http://student.csc.liv.ac.uk/cgi-bin/cgiwrap/sh/demo/more/dirs?

# QUERY_STRING
username=dave&fullname=David+Davidson

# REQUEST_METHOD
GET

# PATH_INFO
/more/dirs

# PATH_TRANSLATED
/usr/www/externals/docs/more/dirs

# SCRIPT_NAME
/cgi-bin/cgiwrap/sh/demo

# SCRIPT_FILENAME
/usr/users/loco/sh/public_html/cgi-bin/demo
```

Python: Basic Syntax

- A Python program/script consists of one or more statements and comments
- One-line comments start with `#` and run to the end of the line
- Multi-line comments consist of several one-line comments
- Statements are delimited by newlines except where a newline is escaped (by a backslash `\`)
- On Unix/Linux systems, Python scripts begin with `#!` (called `hash bang` or `she bang`) and the location of the Python interpreter/compiler

```python
# Our first Python script
# HelloWorld.py

#!/usr/bin/python3

# Hello World
print("Hello World")
```

Environment variables: POST

**Env variable** | **Meaning**
---|---
QUERY_STRING | The query information passed to the program
REQUEST_METHOD | The request method that was used
SCRIPT_NAME | The relative virtual path of the CGI program
SCRIPT_FILENAME | The physical path of the CGI program

```python
POST /cgi-bin/cgiwrap/sh/demo
Host: student.csc.liv.ac.uk

username=2560x4228x=2277& fullname=Peter+Newton

# QUERY_STRING

# REQUEST_METHOD
POST

# SCRIPT_NAME
/cgi-bin/cgiwrap/sh/demo

# SCRIPT_FILENAME
/usr/users/loco/sh/public_html/cgi-bin/demo
```

Python: Basic Syntax

- Strictly speaking, in Python one assigns a (variable) name to a value, not the other way round
- A (variable) name does not exist before the first assignment
- But, the syntax for an assignment is the same as in JavaScript

```python
age = 23
The first assignment to a variable defines that variable

Python supports the standard binary assignment operators

age += 10

Python uses static scoping

Blocks of statements, called suites are delimited with indentation
- each time the level of indentation is increased, a new block starts
- each time the level of indentation is decreased, a block has ended

A colon : separates the header of block from the rest of the suite
```

More environment variables

**Env variable** | **Meaning**
---|---
HTTP_ACCEPT | A list of the MIME types that the client can accept
HTTP_REFERER | The URL of the document that the client points to before accessing the CGI program
HTTP_USER_AGENT | The browser the client is using to issue the request
REMOTE_ADDR | The remote IP address of the user making the request
REMOTE_HOST | The remote hostname of the user making the request
SERVER_NAME | The server’s hostname
SERVER_PORT | The port number of the host on which the server is running
SERVER_SOFTWARE | The name and version of the server software

Python: Type System

- Python is a dynamically typed language:
  - a variable declaration does not include a type and
  - a variable can hold values of different types over time
  ```python
  x = "Hello"
  x = 42
  ```
  - is a valid sequence of statements
- Python is (mostly) strongly typed language:
  - values are not automatically converted from ‘unrelated’ types
  ```python
  y = "Hello" + 42
  ```
  - will cause an error in Python
- However, quite a number of types are considered to be ‘related’
  ```python
  z = 42 and True
  ```
  - will not cause an error in Python although a boolean operator is applied to a number
Python: Conditional Statements

- A string literal is a sequence of characters surrounded by single-quotes, double-quotes, or triple-quotes
  "chars" single-quoted string
  "chars" double-quoted string
  '''chars''' triple-quoted string, can span several lines and contain single and double quotes, but not at the start or end
  """This is a triple-quoted "string" containing "quotes" and spanning more than one line""
- In all these forms \\ acts as escape character

Python: Modules

- A lot of functionality of Python is contained in modules
- The statement
  
  ```python
  import module1, module2, ..., moduleN
  ```
  
  makes all functions from modules module1, module2 available, but all function names must be prefixed by the module name
  ```python
  import math
  print math.factorial(5)
  ```
- The statement
  ```python
  from module1 import fun1[, fun2[, ... funN]]
  ```
  
  imports the named functions from module module1 and makes them available without the need for a prefix
  ```python
  from math import factorial
  print factorial(5)
  ```

Python: The re module

The `re` module of Python provides functions that use regular expressions

```python
re.match(regex,string [,flags])
```

attempts to find a match for `regex` at the start of `string`
returns a match object when `regex` is found and `None` otherwise

```python
re.match(regex, string [,flags])
```

attempts to find a match for `regex` anywhere in `string`
returns a match object when `regex` is found and `None` otherwise

Python: Type System: Dictionaries

- A Python dictionary is a mapping of keys to values
  (aka associative array or hash table)
- A dictionary literal is a comma-separated list of key-value pairs consisting of a `key` and a `value` separated by a colon:
  surrounded by curly brackets
  ```python
  {'name': 'Dave', 'age': 23, 'height': '185cm'}
  ```
- Elements of any immutable type, e.g. strings, can be used as keys
- The value associated with a specific key in a dictionary dict can be accessed (and modified) using
  ```python
  dct[key]
  ```

Python: Type System: Strings

- A string literal is a sequence of characters surrounded by single-quotes, double-quotes, or triple-quotes
  'chars' single-quoted string
  "chars" double-quoted string
  '''chars''' triple-quoted string, can span several lines and contain single and double quotes, but not at the start or end
  """This is a triple-quoted "string" containing "quotes" and spanning more than one line""

Python: Functions

Functions are elements of type `function` and can be defined as follows:

```python
def identifier(param1,param2,...):
  docstring
```

- The function name `identifier` is case-sensitive
- The function name must be followed by parentheses
- A function has zero, one, or more parameters that are variables
- Parameters are not typed
- `docstring` is a string describing the function and will be returned by `help(identifier)` or `identifier.__doc__`
- `suite` is a non-empty sequence of statements

A function is called by using the function name followed by a list of arguments in parentheses

```python
... identifier(arg1, arg2,...) ... # Function call
```
The module os provides the environ dictionary
The environ dictionary maps a script’s environmental variables as keys to the values of those variables

```
# Make sure output uses UTF-8 encoding
sys.stdout = codecs.getwriter("utf-8") (sys.stdout.detach ( )

start_html("Where are you coming from?")
user_ip = os.environ["REMOTE_ADDR"]
print("<div>Clients IP address : ' + user_ip + '</div>)")
print("<p>Welcome, university user</p>");
print("<p>Lists more content only available to university users</p>");
else:
    print("<div>Sorry, please come back when you are on a uni computer</div>)")
end_html()
```

The module cgi provides methods to access the input data of HTML forms in a two step process:

1. Create an instance of the FieldStorage class and assign it to a variable

```
variable = cgi.FieldStorage()
```

This reads the form data from standard input or the environment variable QUERY_STRING
2. If there is a FieldStorage object:
   - `variable['name'].value`
   - `variable.getvalue('name')`
   - `variable.getfirst('name',default=None)`
   - `variable.getlist('name')`


```
# The form
<form method="POST">
  <label>User name : <input type="text" name="user" /></label>
  <label> Email address : <input type="text" name="email" /></label>
  <input type="submit" name="submit" />
</form>
```

We want to create a CGI script that both creates the following form and validates its inputs

```
def printForm():
    print("<form method="POST">
          <label>User name : <input type="text" name="user"></label>
          <label>Email address : <input type="text" name="email"></label>
          <input type="submit" name="submit" />
    </form>")
```

```
def validateName(field):
    if field == "" or field == None:
        err = "No name entered\n"
    elif len(field) < 5:
        return "Invalid character in name\n"
    else:
        return ""
```

```
def validateEmail(field):
    if field == "" or field == None:
        return "No email entered\n"
    elif not ((field.find("@") > 0) and (field.find(".")) > 0)):
        return "Symbol @ or . missing\n"
    elif re.search(r'[^a-zA-Z0-9_\-\.]', field):
        return "Invalid character in email\n"
    else:
        return ""
```


For information on modules for CGI programming in Python see
\- cgi: Common Gateway Interface support
\- os: Miscellaneous operating system interfaces

PyCon.org, 29 October 2019. https://docs.python.org/3/library/cgi.html
https://docs.python.org/3/library/os.html