	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
9:00 - 10:00					
10:00 - 11:00			Consultation ICT3.20	INFT1004 Lab 4 *ICT3.44* Will	
11:00 - 12:00			INFT1004 Lab 1 - BYOD *ICT3.29* Keith		
12:00 - 1:00				INFT1004 Lab 5 *ICT3.44* Will	
1:00 - 2:00			PASS MCG 29		
2:00 - 3:00		PASS W 238	INFT1004 Lab 2 *ICT3.37* Brendan		
3:00 - 4:00		INFT1004 Lecture GP 201			
4:00 - 5:00			INFT1004 Lab3 *ICT3.44* Brendan	INFT1004 Lab 6 *ICT3.44* Will	
5:00 - 6:00					
6:00 - 7:00					
7:00 - 8:00					

INFT1004 - SEMESTER 1 - 2017		LECTURE TOPICS	
Week 1	Feb 27	Introduction, Assignment, Arithmetic	
Week 2	Mar 6	Sequence, Quick Start, Programming Style	
Week 3	Mar 13	Pictures, Functions, Media Paths	
Week 4	Mar 20	Arrays, Pixels, For Loop, Reference Passing	
Week 5	Mar 27	Nested Loops, Selection, Advanced Pictures	~~Practical Test~~
Week 6	Apr 3	Lists, Strings, Input & Output, Files	Practical Test
Week 7	Apr 10	Drawing Pictures, Program Design, While Loop	Assignment set
Recess	**Apr 14 – Apr 23**	**Mid Semester Recess Break**	
Week 8	Apr 24	No Lecture / Revision and Assignment in Labs	
Week 9	May 1	Data Structures, Processing sound	
Week 10	May 8	Advanced sound	Assignment part 1 due 8:00am Tue, May 9
Week 11	May 15	Movies, Scope, Import	
Week 12	May 22	Turtles, Writing Classes	Assignment part 2 due 8:00am Tue, May 23
Week 13	May 29	Revision	
Mid Year Examination Period - MUST be available normal & supplementary period			

Lecture Topics and Lab topics are the same for each week

INFT1004

Introduction to Programming

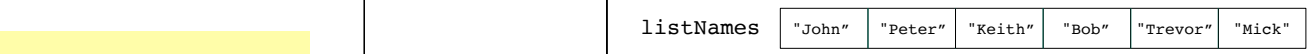
Module 6.1
Introduction to Lists

Guzdial & Ericson - Third Edition – chapters 10 and 11
Guzdial & Ericson - Fourth (Global) Edition – chapters 11 and 12

Introducing Lists

Lists are a type of data structure

They allow you to store a number of elements together



A list of strings.

(Strings are one type of data used in python – they act like simple types)

Introducing Lists

Lets look at this simple code that uses a list.

```
def listExampleSimple():
   listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
   print listNames
   print listNames[3]
```


Mod6_1_ListExamples.py

List of Names

I called my list - listNames

JES doesn't much care what I call it – but this is good for me as I'm going to use it to hold a list of names.

```
def listExampleSimple():
   listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
   print listNames
   print listNames[3]
```


Example – List of Names

It contains 6 elements – all strings

The strings are separated by commas.

I could make it longer or shorter

```
def listExampleSimple():
   listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
   print listNames
   print listNames[3]
```


Example – List of Names

We can print the list.

```
def listExampleSimple():
   listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
   print listNames
   print listNames[3]
```

Example – List of Names

We can print one of the list elements

Try it

Which string does it print?

```
def listExampleSimple():
   listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
   print listNames
   print listNames[3]
```


Another Example – More Lists

Lets change the program to add some extra bits

```
def listExampleMore():
   listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
   print listNames
   print listNames[3]

   # get the number of elements in the list - print this out
   numberElements = len(listNames)
   print ("number of items = " + str(numberElements))

   #try this special function with the list
   specialMessage(listNames)
```

Mod6_1_ListExamples.py

Another Example – More Lists

numberElements is a variable – variables can change in value. I made this name up because it makes sense to me – I'm going to use this variable to store the number of elements in the list.

```
def listExampleMore():
   listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
   print listNames
   print listNames[3]

   # get the number of elements in the list - print this out
   numberElements = len(listNames)
   print ("number of items = " + str(numberElements))

   #try this special function with the list
   specialMessage(listNames)
```


Another Example – More Lists

This statement uses the "len" function (declared in Python) to find out how many elements are in the list.

```
def listExampleMore():
   listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
   print listNames
   print listNames[3]

   # get the number of elements in the list - print this out
   numberElements = len(listNames)
   print ("number of items = " + str(numberElements))

   #try this special function with the list
   specialMessage(listNames)
```

Another Example – More Lists

numberElements is an integer.

```
def listExampleMore():
    listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
    print listNames
    print listNames[3]

    # get the number of elements in the list - print this out
    numberElements = len(listNames)
    print ("number of items = " + str(numberElements))

    #try this special function with the list
    specialMessage(listNames)
```


Another Example – More Lists

The len function returns an integer. This is important because you need to have both sides of the assignment with the same type.

```
def listExampleMore():
    listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
    print listNames
    print listNames[3]

    # get the number of elements in the list - print this out
    numberElements = len(listNames)
    print ("number of items = " + str(numberElements))

    #try this special function with the list
    specialMessage(listNames)
```


Another Example – More Lists

The len function returns an integer. This is important because you need to have both sides of the assignment with the same type. (More theory next week).

```
def listExampleMore():
    listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
    print listNames
    print listNames[3]

    # get the number of elements in the list - print this out
integer          = integer
    print ("number of items = " + str(numberElements))

    #try this special function with the list
    specialMessage(listNames)
```


Another Example – More Lists

The len function requires one argument – which needs to be a list. (Because whoever wrote the function decided that's what they wanted).

```
def listExampleMore():
    listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
    print listNames
    print listNames[3]

    # get the number of elements in the list - print this out
    numberElements = len(listNames)
    print ("number of items = " + str(numberElements))

    #try this special function with the list
    specialMessage(listNames)
```

Another Example – More Lists

This line prints the number of items by first concatenating two strings.

```
def listExampleMore():
    listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
    print listNames
    print listNames[3]

    # get the number of elements in the list - print this out
    numberElements = len(listNames)
    print ("number of items = " + str(numberElements))

    #try this special function with the list
    specialMessage(listNames)
```


Another Example – More Lists

But because numberElements is an integer we need to turn it into a string first. The str function does this.

```
def listExampleMore():
    listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
    print listNames
    print listNames[3]

    # get the number of elements in the list - print this out
    numberElements = len(listNames)
    print ("number of items = " + str(numberElements))

    #try this special function with the list
    specialMessage(listNames)
```


Another Example – More Lists

print, len and str are all functions that come with JES / Python.

The last line calls a function that I wrote !

```
def listExampleMore():
    listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
    print listNames
    print listNames[3]

    # get the number of elements in the list - print this out
    numberElements = len(listNames)
    print ("number of items = " + str(numberElements))

    #try this special function with the list
    specialMessage(listNames)
```


Another Example – More Lists

My function has a name – specialMessage (I made this name up)

It has one argument (listNames)

```
def listExampleMore():
    listNames = ["John","Peter","Keith","Bob","Trevor","Mick"]
    print listNames
    print listNames[3]

    # get the number of elements in the list - print this out
    numberElements = len(listNames)
    print ("number of items = " + str(numberElements))

    #try this special function with the list
    specialMessage(listNames)
```

Another Example – More Lists

Here is my function – it has one parameter – I decided to called this parameter - aNameList

```
def specialMessage(aNameList):
    #This function looks for a particular string in a list of a names
    #If it finds that name it prints a special message

    for item in aNameList :  #process each item in the list - iteratio
       if item == "Keith" :  #only print the special case - selection
          print "Happy Birthday " + item + " - you are a legend!"
```

Mod6_1_ListExamples.py

Another Example – More Lists

Here is my function – it has one parameter – I decided to called this parameter - aNameList

```
def specialMessage(aNameList):
    #This function looks for a particular string in a list of a names
    #If it finds that name it prints a special message

    for item in aNameList :  #process each item in the list - iteratio
       if item == "Keith" :  #only print the special case - selection
          print "Happy Birthday " + item + " - you are a legend!"
```

```
specialMessage(listNames)
```

When I call the function the number and type of arguments needs to match the number and type of parameters (the names don't need to be the same)

Another Example – More Lists

It uses iteration – Processing every element in the list using a for loop

```
def specialMessage(aNameList):
    #This function looks for a particular string in a list of a names
    #If it finds that name it prints a special message

    for item in aNameList :  #process each item in the list- iteration
       if item == "Keith" :  #only print the special case- selection
          print "Happy Birthday " + item + " - you are a legend!"
```

This is just one way of doing iteration – we can also use while loops etc.

Another Example – More Lists

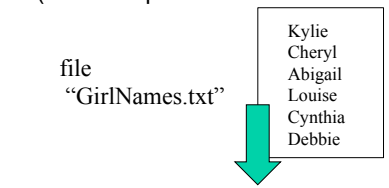
It uses selection – if it finds an element in the list that has the value, "Keith" it will print the special message.

```
def specialMessage(aNameList):
    #This function looks for a particular string in a list of a names
    #If it finds that name it prints a special message

    for item in aNameList :  #process each item in the list - iteratio
       if item == "Keith" :  #only print the special case - selection
          print "Happy Birthday " + item + " - you are a legend!"
```

Even More Lists (and Files)

Lets get some data from a file – we will use lists again

We read some names from a file, store them in a list (and then print them from the list.)




```
nameList = ["Kylie","Cheryl","Abigail","Louise","Cynthia","Debbie"]
```


Mod6_1_ListExamples.py

A file example

Files are one way to get information in and out of your program – in this case we will use a text file – you can create your own in a text editor or even excel.

I've made one called "GirlNames.txt" you can try


```
def readPrintFile():
   #This is a function that allows the user to pick a text file that
   #contains a lot of names (one name on each line).
   #It places the names into a list. It then prints each element of the list.

   #1. Pick a text file that contains names - one name on each line
   filename = pickAFile()

   #2. Open the file for reading
   file = open(filename, "r")

   #3. Copy all the names into the list
   nameList =[] #start with an empty list

   for line in file:
      nameList.append(line)

   print("nameList = " + str(nameList)) #print the list just to check

   #4. Print all the names in the list – one at a time
   for name in nameList:
      print name

   #5. Close your file at the end
   file.close()
```

Mod6_1_FileReadExamples.py


```
def readSortedList(pathName):

   #1. Open the file for reading
   file = open(pathName, "r")

   #2. Copy all the names into the list
   nameList =[] #start with an empty list

   # process each line in the file - one at a time
   for line in file:
      characters = len(line)  # work out how many characters in the line  - the length

      # check the last character in the string in case it is a newline "\n"
      if line[characters-1] == "\n":
         removedNewline =  line[0:characters-1]  #don't use the last character ("/n
      else:
         removedNewline =  line[0:characters]    #keep the last character ("/n")

      nameList.append(removedNewline)  # add the item to the end of the list

   #print the list just to check what it looks like before sorting
   print("nameList = " + str(nameList))

   .
   .
   .
```

Mod6_1_FileReadExamples.py

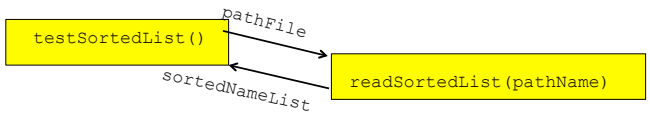
```
def readSortedList(pathName):

.
.
.

   #3. Create the sorted list - we first need to copy the original list
   #   I want to avoid changing this original list - the python sort() function'
   #   has this side effect - so first make a copy in a new list
   sortedList = [] # start with an empty list

   # now copy every name into this new list –
   # if this is confusing try printing the list on each loop
   for name in nameList:
      sortedList.append(name)
      # print sortedList

   # now sort it - this is easy as python has a function that works with lists
   # and does some sorting - for strings it sorts based on the first character of s
   sortedList.sort()

   #4. Close your file at the end
   file.close()

   #5. return the sorted List
   return sortedList
```

Mod6_1_FileReadExamples.py


```
def testSortedList():
   # this function is used to test the readSortedList function
   # Note: You should use setMediaPath() first to make sure the media path
   # is set to the directory where the file called GirlNames.txt is

   # set up the full path name (make sure you have set the media path!)
   pathFile = getMediaPath("GirlNames.txt")

   # read the sorted list
   sortedNameList = readSortedList(pathFile)

   # print the list to check it is correct
   # - notice the str function works with lists
   print "sortedNameList = " + str(sortedNameList)
```

Mod6_1_FileReadExamples.py


```
def testSortedList():
   # this function is used to test the readSortedList function
   # Note: You should use setMediaPath() first to make sure the media path
   # is set to the directory where the file called GirlNames.txt is

   # set up the full path name (make sure you have set the media path!)
   pathFile = getMediaPath("GirlNames.txt")

   # read the sorted list
   sortedNameList = readSortedList(pathFile)

   # print the list to check it is correct
   # - notice the str function works with lists
   print "sortedNameList = " + str(sortedNameList)
```

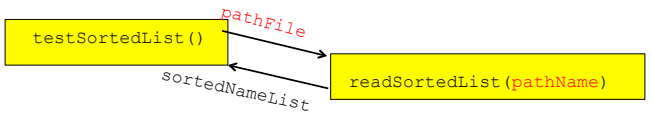


Notice how the first function passes information into the second using an argument - The second function passes information back using the "return" statement - The first function saves this returned information in a variable so it can use it


```
def testSortedList():

   # read the sorted list
   sortedNameList = readSortedList(pathFile)
```

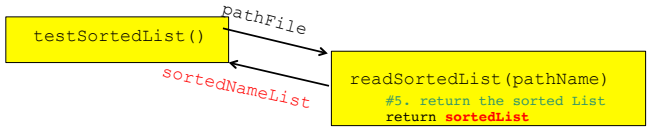
```
def readSortedList(pathName)
```

Notice the name of the argument does not need to match the name of the parameter (Python works it all out – but not based on names)



```
def testSortedList():

    # read the sorted list
    sortedNameList = readSortedList(pathFile)
```

```
def readSortedList(pathName)

    #5. return the sorted List
    return sortedList
```



Notice the name of the returned things also does not need to match the name of the variable you save the return thing in (Python works it all out – but not based on names)

INFT1004

Visual Programming

Module 6.2
More about Strings

Guzdial & Ericson - Third Edition - chapters 9 and 10
Guzdial & Ericson - Fourth (Global) Edition – chapters 10 and 11

Revision
Strings

A string is a sequence of characters

We've denoted literal strings by enclosing the characters in quotation marks:

```
"This is a string"
```

We can also use apostrophes:

```
'Another string'
```


Revision
Special characters in strings

Strings can include invisible characters such as tab, backspace, and enter/return

To include these in a literal string, we can use the special symbols \t, \b, and \n

```
"A string with \ttab and \n\n\nline breaks"
```

Revision

Special characters in strings

A backslash tells Python that the next character is to be treated as a special code

If we want a backslash to be taken literally, we write **r** in front of the string . . .

```
filePath = r"C:\uni\Inft1004\PythonPrograms"
```


Revision

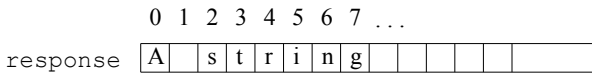
Strings and Lists are alike

A string is stored like an array of characters, with each character having its own index

```
response[0]   is  "A"

response[5]   is  "i"
```


Revision

Strings and Lists are alike



A substring can be extracted using a range of indexes:

```
response[2:6]    is  "stri"
response[:6]     is  "A stri"
response[6:]     is  "ng"
```


Revision

Concatenating +

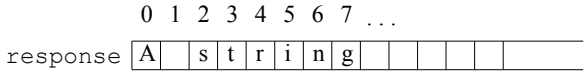
`"Good" + "Day"`	`"GoodDay"`
`"height=" + str(2.3)`	`"height=2.3"`

Revision

Length of a string or list – len(..)

```
len(response)
```

8



More about String methods

Many of the string functions that come with python use dot notation

Dot notation is a common notation in object-oriented programming – (string is actually a class)

This will mean that you call these functions slightly differently.

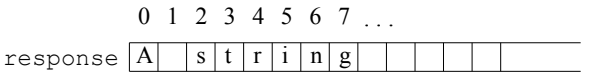
String methods use dot notation

To use such a method, we write the object, then a dot, then the method name, then parentheses (either empty or with the required arguments)

```
response.startswith("A s")
```

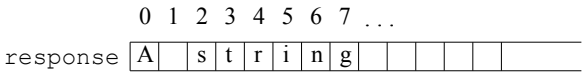
string object — dot — method — parentheses — arguments

startswith() and endswith()



`startswith(str)` returns `true` if the string starts with `str` and `false` if it doesn't

startswith() and endswith()



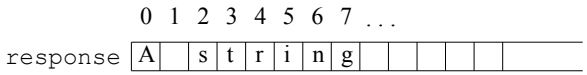
`startswith(str)` returns `true` if the string starts with `str` and `false` if it doesn't

```
print response.startswith("A s")
```
will print 1 (true)

```
print response.startswith("As")
```
will print 0 (false)

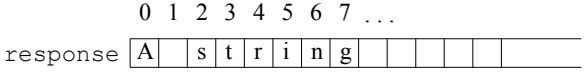
(remember Python uses 1 for true and 0 for false)

startswith() and endswith()



`endswith(str)` returns `true` if the string ends with `str` and `false` if it doesn't

startswith() and endswith()



`endswith(str)` returns `true` if the string ends with `str` and `false` if it doesn't

```
print response.endswith("bing")
```
will print 0 (false)

```
print response.endswith("ring")
```
will print 1 (true)

(remember Python uses 1 for true and 0 for false)

Splitting a String

Splits a string into a list of strings - based on a separation string

```
data = "1.2|2.3|4.3|8.1"

dataList = data.split("|")

# dataList will be a list of
# the following strings
# ["1.2", "2.3", "4.3", "8.1"]
```

Splitting a String

Splits a string into a list of strings - based on a separation string

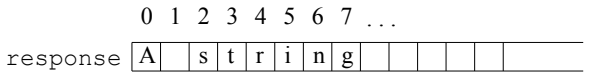
```
data = "1.2|2.3|4.3|8.1"

dataList = data.split("|")

# dataList will be a list of
# the following strings
# ["1.2", "2.3", "4.3", "8.1"]
```

Handy function for breaking up a comma delimited string – or even a space delimited string ;-)

More string methods

`response.find(str)`

return the index at which `str` starts in `response`, or –1 if `str` isn't found in `response`




```
print response.find("ri")    #prints 4
print response.find("XX")    #prints -1
```


More string methods

`response.find(str)`

return the index at which `str` starts in `response`, or –1 if `str` isn't found in `response`

`response.find(str, start)`

similar, but looks only from index `start` onward

`response.find(str, start, finish)`

similar, but looks only between indexes `start` and `finish`

More string methods

`response.upper()`

return `response` all in upper case

`response.lower()`

return `response` all in lower case

`response.title()`

return `response` all in title case
(capital letter at the start of each word)

`response.swapcase()`

return `response` with the case of all letters swapped

Still more string methods

`response.isalpha()`

returns true if `response` consists only of letters, false otherwise

`response.isdigit()`

returns true if `response` consists only of digits, false otherwise

`response.replace(oldStr, newStr)`

returns `response` with all instances of `oldStr` replaced with `newStr`

Still more string methods

There are more string functions.

If you need one and it's not listed here or in the book, take a look online and see if you can find it.

INFT1004

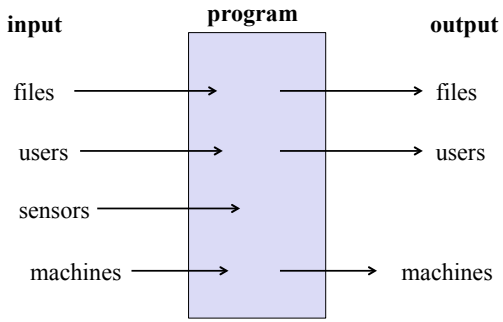
Introduction to Programming

Module 6.3
Input and Output

Guzdial & Ericson - Third Edition – chapter 10
Guzdial & Ericson - Fourth (Global) Edition – chapter 11

Input and Output

Many programs allow for some kind of input and output of data/information



Input and Output - Files

JES provides us with some quite powerful tools for working with files – both for input and output

We will work with quite a few different types of files in this course. These include:

pictures (png, jpg)
sounds (wav)
text files (.txt, .csv)
movies (.avi)

Input and Output - Files

JES provides us with some quite powerful tools for working with files – both for input and output

We will work with quite a few different types of files in this course. These include:

pictures (png, jpg)
sounds (wav)
text files (.txt, .csv)
movies (.avi)

With JES this is the easiest and most powerful way we have of doing input and output

Input and Output - Files

JES provides us with some quite powerful tools for working with files – both for input and output

We will work with quite a few different types of files in this course. These include:

pictures (png, jpg)
sounds (wav)
text files (.txt, .csv)
movies (.avi)

Assignment

Input – Text Files

Sometimes working with text files you get some strange characters.

For example, newlines "\n" and carriage returns "\r" – even extra spaces that might be annoying

You may even find differences between files created on Macs and Windows – so it can be a bit tricky

Input – Text Files

These string functions can be useful…

```
# remove newlines
newString = aString.replace("\n"," ")

# remove carriage returns - sometimes in windows
newString = aString.replace("\r"," ")

# remove tabs
newString = aString.replace("\t"," ")

# remove trailing and leading white space
newString = aString.strip()
```


Input and Output - User

Many programs provide sophisticated user interfaces that allow the user to complete complex tasks using a purpose built visual interface.

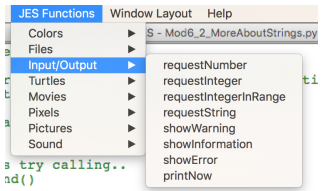
This is possible with python and JES using java – but it is not a simple task – and beyond what you will learn in this course

Input and Output - User

We will work with some simple user interfaces that take some basic text input from the user and produce basic output – again as text.

We will use predefined JES functions like:

- requestString
- requestInteger
- requestNumber
- showInformation
- print
- printNow



Input - requestString()

`requestString()` returns a string value

It takes one argument – a string message

Mod6_3_testInputOutput.py

Help about requestString()

Check JES for more help about JES functions!

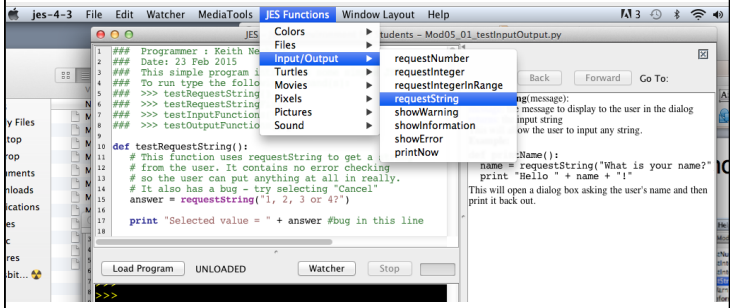
requestString(message):
message: the message to display to the user in the dialog
returns: the input string
This will allow the user to input any string.

Example:

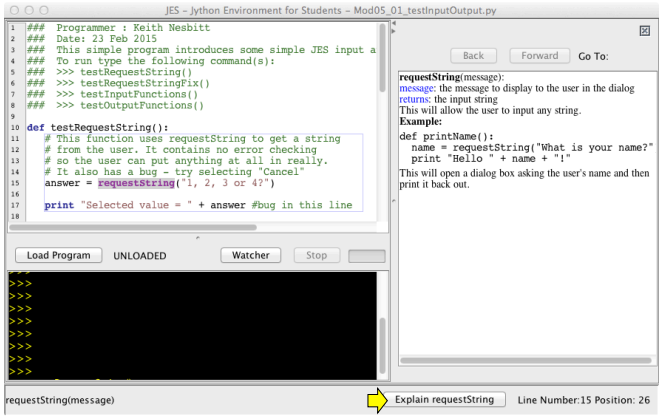
```
def printName():
  name = requestString("What is your name?")
  print "Hello " + name + "!"
```

This will open a dialog box asking the user's name and then print it back out.

Help with JES functions



Try selecting the function `requestString` function from JESFunctions menu to get help



Or try selecting the function name `requestString` in your code and then use the "Explain" button in JES

More Input in JES

```
requestNumber(message)

requestInteger(message)

requestIntegerInRange(message, min, max)
```

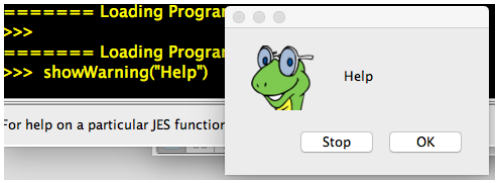
More Input in JES

```
def testInputFunctions():
   #This functions tests some useful JES input functions
   decimalValue = requestNumber("Enter a decimal number Will")
   printNow(decimalValue)

   integerValue = requestIntegerInRange("Enter an integer 1-5", 1, 5)
   printNow(integerValue)

   integerValue = requestInteger("Enter an integer Will")
   printNow(integerValue)
```

Mod6_3_testInputOutput.py

Some Output in JES

```
showWarning("Help")
showInformation(message)
showError(message)
```



Some Output in JES

```
def testOutputFunctions():
   #This functions tests some useful JES output functions

   showWarning("Warning Warning Will Robinson")

   showInformation("Will is lost in space")

   showError("I'm afraid I can't do that Will")
```

Mod6_3_testInputOutput.py

Getting user input

So far all of our programs have run without input from the user

The text-based adventure game in chapter 10 of the textbook illustrates the use of `requestString()` to get user input

This chapter als0 goes a lot more into string processing

INFT1004

Visual Programming

Module 6.4
Files

Guzdial & Ericson - Third Edition – chapter 10
Guzdial & Ericson - Fourth (Global) Edition – chapter 11

Processing Files

Files are the basic unit of storage on computers

Some of the things stored in files are:

- word-processing documents (eg .doc)
- text files (eg .txt)
- web pages (eg .htm)
- spreadsheets (eg .xls)
- comma-separated values files (eg .csv)
- program files (eg .py)
- pictures (eg .jpg)
- sounds (eg .wav)

Text files

Text files are easily interpreted as strings

We can process them with the string methods that we've now learnt

For example, we can read the text from a file, convert it all to title case, and write it back to the file

Or we can read the text from a file, substitute one string for another wherever it occurs, and write the file back

Opening a file

Opening a file

`open(filename, "rt")` – open a file to read as text
`open(filename, "wt")` – open a file to write as text
`open(filename, "at")` – open a file to append text
`open(filename, "rb")` – open a file to read as bytes
`open(filename, "wb")` – open a file to write as bytes

Mod6_4_Files.py

Opening a file

In each case, *filename* can be . . .

A full path and filename

`getMediaPath(filename)`

A variable with the filename and path in it
For example obtained from

```
filename = pickAFile()
```


Text file operations

`file.read()`	Read the whole file as a single string
`file.readline()`	Read the next line from the file as a string
`file.readlines()`	Read the file as a list of strings, one for each line

Note that reading a file consumes it; what you've read from the file is removed from the file object – though the file from which it was opened is still untouched

Mod6_4_Files.py

Text file operations

`file.write(string)`

– adds the string to the end of the file

Note that when you open a file for writing, it's considered empty, so anything you add to it is added to an initially empty file

Mod6_4_Files.py

Closing a file

It's very important to close a file when you've finished the current operations on it

`file.close()`

When writing a file, the content is buffered: it's actually written to the file not whenever the program says write, but when there's enough to be worth writing

Closing a file

If your program finishes while files are still open, the last buffer might not be written

Closing the file ensures that the last buffer will be written

This is easy to overlook because it's not always obvious: you can sometimes get away with forgetting.

Be alert!

reading a text file as a string

```
def readFileText():
   # This function reads a file of text from
   # a file as a single string and prints it

   # pick a file for opening
   filename = pickAFile()

   # open the text file for reading
   file = open(filename, "rt")

   # read the text as a single string
   stringText = file.read()

   file.close()           # always close the file

   print stringText       # print the string
```


Common mistakes with files

1. Trying to read a file that's opened for write
2. Trying to write a file that's opened for read
3. Trying to read/write a file that you've closed
4. Forgetting to close a file when you've finished with it
5. Failing to close a read file before opening it to write

Common mistakes with files

6. Forgetting that reading consumes a file
7. Opening a file to write when you meant to open it to append
8. Failing to specify the right directory for the file
9. Overwriting an existing file – Python doesn't automatically warn you before doing this

What to do this week

- [ ] Check the Lab for Week 6 (this might help with assignment)
- [ ] Do the Practical Test for Week 6
- [ ] Mark the Practical Exam (or get a peer to do it)
- [ ] Problems with your Practical test – next weeks tutorial!
- [ ] No quiz this week
- [ ] Keep reading the textbook