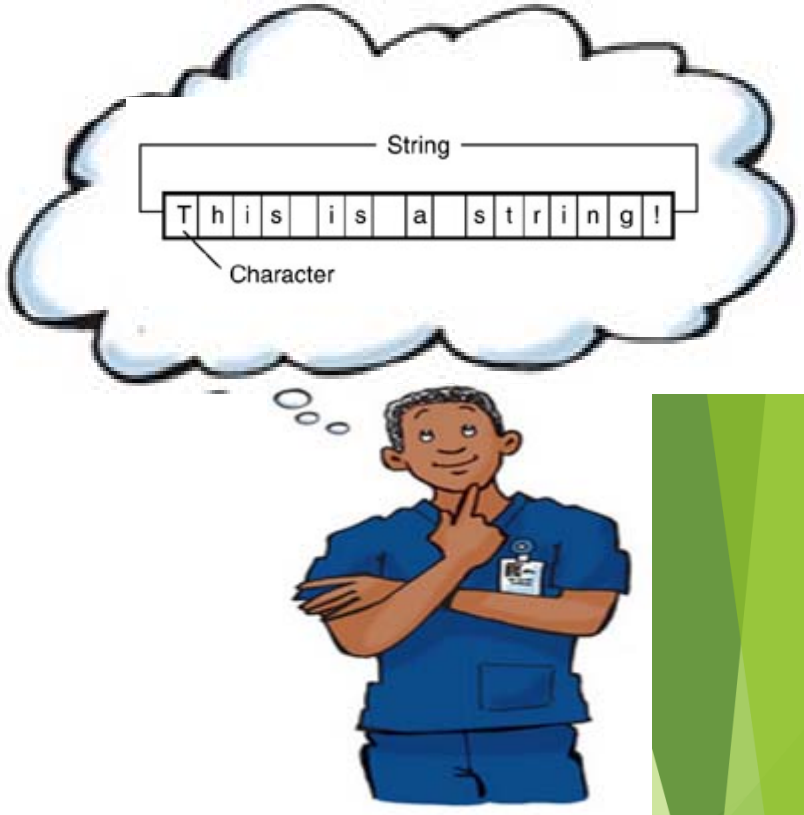
String Operations



Introduction to Programming

This Lecture

- Refreshing our knowledge about String
  - String Literals
  - “Type” function
  - Operators and Expressions
- Indexing a String
- Slicing a String
- Length of a String
- String Methods
- Many Examples

String Literals (1)

String literals contain portions of text within code.

They must be enclosed in matching quote characters, either single (') or double ("). The following are all legal:

```
"Good morning. How are you?"
'Good morning. How are you?'
"Tony's bicycle"
'He said, "Hello!"'
```

String Literals (2)

The backslash \ character has a special role; it denotes the start of an ***escape sequence***. Escape sequences are used to allow strings to include characters that would otherwise cause a problem:

`\'`	Single quote	`\"`	Double quote
`\n`	Newline	`\t`	Tab

see http://docs.python.org/reference/lexical_analysis.html section 2.4 for more

Printing Strings

Use the print statement to print a string

```
print "Good morning.\nHow are
  you?"
Good morning.
How are you?
```

By default print adds a newline character at the end to terminate the line.

```
print "Hello
  "
print
  "World"
Hello
World
```

Putting a trailing comma at the end of a print statement suppresses this behaviour.

```
print "Hell
  ",
print
  "World"
Hello World
```

Examples - the type function

```
>>> x = "3.14159"
>>> print type(x)
<type 'str'>
>>> y = 3.14159
>>> print type(y)
<type 'float'>
>>> z = 3
>>> print type(z)
<type 'int'>
```

Operators and Expressions

Strings (and other sequences covered later) can be concatenated by the addition operator (+). The operator * may be used to concatenate multiple copies of a string (or other sequence), e.g.

```
line = 30*"="
print line
town = raw_input("Where do you live? ")
print "You live in " + town + "."
print line
```

```
==============================
Where do you live? Singapore
You live in Singapore.
==============================
```

Before using the operators " * " and "+", check whether data type of the variables is string.

An **expression** is a phrase of code that the interpreter can evaluate to produce a value. The simplest expressions are literals and variables. More complicated expressions are built by joining literals and/or variables with operators. An expression may be used anywhere instead of a variable or literal (except on the left hand side of assignment statement).

Indexing a String

The first character of a string `s` can be accessed as `s[0]` (also a string).
`s[1]` and `s[2]` are the 2nd and 3rd characters, and so on.
`s[-1]` and `s[-2]` are the last and penultimate characters respectively.

```
name=raw_input("What's your name? ")
print "First letter is " + name[0]
print "Last letter is " + name[-1]
```

```
What's your name? Thomas
```

```
First letter is T
Last letter is s
```

Slicing a String

If `s` is a string then `s[i:j]` is the substring of `s` from the *i*th element, included, to the *j*th element, excluded (specifying bounds in this way is standard practice in Python).

```
name="Don Quijote"
print name[4:8]
```

Quij

Length of a String

The built-in function `len` returns the number of characters in a string, e.g.

```
s="hello"
print len(s)
```

```
5
```

String Methods: Length of a String

- **Methods** are functions specifically associated with a particular kind of **object**. Now we see strings are objects

```
s="hello"
print s.__len__()
```

```
5
```

String Methods

- There are many string methods; type “help(str)” in Python shell to see all methods.
- here are just a few, as they would be applied to a string stored in “s” .
  - s.lower()
    - Returns a copy of s with all letters converted to lowercase.
  - s.upper()
    - Returns a copy of s with all letters converted to uppercase.
  - s.count(sub)
    - Returns the number of occurrences of substring sub in s.
  - s.replace(old,new)
    - Returns a copy of s with the substring old replaced by the string new.
    - e.g. "cap".replace("a","u") is "cup".
  - s.strip(x)
    - Called without an argument, s.strip() returns a copy of s with both leading and trailing whitespace removed. Called with a string argument, it removes any leading or trailing characters contained in string x. Thus s.strip(" \n\r\t") is the same as to s.strip().

Examples :
s.lower() and s.upper()

```
person_name=raw_input("What is your name")

if person_name=="bob" or person_name=="Bob" or person_name=="BOB":
    print " hi Bob"
else:
    print " sorry I don't know you"
```

```
person_name=raw_input("What is your name")
pn=person_name.upper()
if pn=="BOB":
    print " Hi Bob"
else:
    print " sorry I don't know you"
```

```
person_name=raw_input("What is your name")
pn=person_name.lower()
if pn=="bob":
    print " Hi Bob"
else:
    print " sorry I don't know you"
```

Examples :
s.count(sub)

```
person_name=raw_input("What is your name ?")
pn=person_name.lower()

number_of_a=pn.count("a")
number_of_e=pn.count("e")
number_of_i=pn.count("i")
number_of_o=pn.count("o")
number_of_u=pn.count("u")
total_number_of_vowels=number_of_a+number_of_e+number_of_i+number_of_o+number_of_u

if total_number_of_vowels>0:
     print " Number of vowels in your name is ",total_number_of_vowels
else:
    print " That is a very strange name . Your name has no vowels"
```

Examples :

```
person_name=raw_input("What is your name ?")
pn=person_name.lower()

number_of_a=pn.count("a")
number_of_e=pn.count("e")
number_of_i=pn.count("i")
number_of_o=pn.count("o")
number_of_u=pn.count("u")
total_number_of_vowels=number_of_a+number_of_e+number_of_i+number_of_o+number_of_u

if total_number_of_vowels>0:
     print " Number of vowels in your name is ",total_number_of_vowels
else:
    print " That is a very strange name . Your name has no vowels"
```

```
vowels="aeiou"
person_name=raw_input("What is your name ?")
pn=person_name.lower()

total_number_of_vowels=0

for l in range(0,len(vowels)):
     total_number_of_vowels=total_number_of_vowels+pn.count(vowels[l])


if total_number_of_vowels>0:
      print " Number of vowels in your name is ",total_number_of_vowels
else:
     print " That is a very strange name . Your name has no vowels"
```

Examples : s.replace(old,new)

```
person_name=raw_input("What is your name ?")
pn=person_name.lower()

pn=pn.replace("one","1")
pn=pn.replace("won","1")
pn=pn.replace("want","1")
pn=pn.replace("too","2")
pn=pn.replace("to","2")
pn=pn.replace("for","2")
pn=pn.replace("ate","8")


if person_name==pn:
     print "Sorry, you can not shorten yoour name"
else:
    print " You can text your name '", pn, "' in this way."
```

Examples :
s.strip(x)

```
#person_name=raw_input("What is your name")
person_name=" Bob "
pn=person_name.lower()
pn=pn.strip()
if pn=="bob":
    print " Hi Bob"
else:
    print " Sorry, I don't know you"
```

Simple example

```
fore    =raw_input("Forename? ")
middle  =raw_input("Middle name? ")
sur     =raw_input("Surname? ")
name = fore[0]+"."+middle[0]+"."+sur
email= name.lower() + "@smu.edu.sg"
print "Name : " + name
print "Email: " + email
```

String Methods (for future reference)

`s.`**`split`**`(sep)` Returns a *list* of "words" from `s`, where string `sep` separates words. e.g.

```
s="oak trees, hazel shrubs"
print s.split(" ")
print s.split(",")
```

```
['oak', 'trees,', 'hazel', 'shrubs']
['oak trees', ' hazel shrubs']
```

`s.`**`join`**`(l)` Returns the string obtained by concatenating the list of strings `l`, inserting `s` as a separator between each item. e.g.

```
l = ["oak","hazel","beech"]
s = "; ".join(l)
print s
```

```
oak; hazel; beech
```

Recommendations

- Hide the complexity in the user defined functions
- Try out the examples in this lecture slides
- Produce alternative solutions

Summary

- Indexing a String
- Slicing a String
- Length of a String
- String Methods
- Structural problem solving examples