Strings in Python

Strings

- A string is an immutable sequence of characters
- A string literal uses quotes
- 'Hello' or "Hello"
- For strings, + means “concatenate”
- When a string contains numbers, it is still a string
- We can convert numbers in a string into a number using int()

String Operations

Operation	Interpretation
`S = ''`	Empty string
`S = "spam's"`	Double quotes, same as single
`S = 's\np\ta\x00m'`	Escape sequences
`S = """...multiline..."""`	Triple-quoted block strings
`S = r'\temp\spam'`	Raw strings (no escapes)
`S1 + S2`	Concatenate, repeat
`S * 3`	
`S[i]`	Index, slice, length
`S[i:j]`	

String Operations

`len(S)`	
`"a %s parrot" % kind`	String formatting expression
`"a {0} parrot".format(kind)`	String formatting method in 2.6, 2.7, and 3.X
`S.find('pa')`	String methods (see ahead for all 43): search,
`S.rstrip()`	remove whitespace,
`S.replace('pa', 'xx')`	replacement,
`S.split(',')`	split on delimiter,

String Operations

Operation	Interpretation
`S.isdigit()`	content test,
`S.lower()`	case conversion,
`S.endswith('spam')`	end test,
`'spam'.join(strlist)`	delimiter join,

Strings - Introduction

- Single quotes: 'spa"m'
- Double quotes: "spa'm"
- Triple quotes: '''...spam ...''', """... spam ..."""
- Escape sequences: "s\tp\na\0m“
- Raw strings: r"C:\new\test.spm"

Escape Sequences

- Represent Special Characters

```
>>> s = 'a\nb\tc'
>>> s
'a\nb\tc'
>>> print(s)
a
b	c
>>> len(s)
5
```

Escape Sequences

Table 7-2. String backslash characters

Escape	Meaning
newline	Ignored (continuation line)
\\	Backslash (stores one \)
\'	Single quote (stores ')
\"	Double quote (stores ")
\a	Bell
\b	Backspace
\f	Formfeed
\n	Newline (linefeed)
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\x*hh*	Character with hex value *hh* (exactly 2 digits)
ooo	Character with octal value *ooo* (up to 3 digits)
\0	Null: binary 0 character (doesn't end string)

String Operations - Length

```
>>> s = 'a\0b\0c'

>>> s

'a\x00b\x00c'

>>>print(s)

a b c

>>> len(s)

5
```

String Operations - Length

- if Python does not recognize the character after a \ as being a valid escape code, it simply keeps the backslash in the resulting string:

```
>>> x = "C:\py\code"
# Keeps \ literally (and displays it as \\)
>>> x
'C:\\py\\code'
>>> len(x)
10
```

Raw Strings Suppress Escapes

- a = 'C:\new\text.dat'
- # consider \n as new line character and \t as tab

```
>>>print(a)
C:
ew     ext.dat
```

Raw string

```
>>> b = r'C:\new\text.dat'
>>> print(b)
C:\new\text.dat
```

Raw Strings Suppress Escapes

- myfile = open('C:\new\text.dat', 'w') # consider
- myfile = open(r'C:\new\text.dat', 'w')
- Alternatively two backslashes may be used
- myfile = open('C:\\new\\text.dat', 'w')

```
>>> path = r'C:\new\text.dat'
>>> path                          # Show as Python code
'C:\\new\\text.dat‘
```

Basic Operations

- >>> 'Ni!' * 4
- # Repetition: like "Ni!" + "Ni!" + ... 'Ni!Ni!Ni!Ni!‘
- >>> print('-' * 80) # 80 dashes, the easy way
- >>> myjob = "hacker"
- >>> for c in myjob:

```
        print(c, end=' ')
    #add end to suppress default new line character
```

h a c k e r

Using ‘in’ Operator in Strings

```
>>> "k" in myjob                    # Found True
>>> "z" in myjob                    # Not found False
>>> 'spam' in 'abcspamdef'
# Substring search, no position returned
True
```

String Operations - Counting

Counting

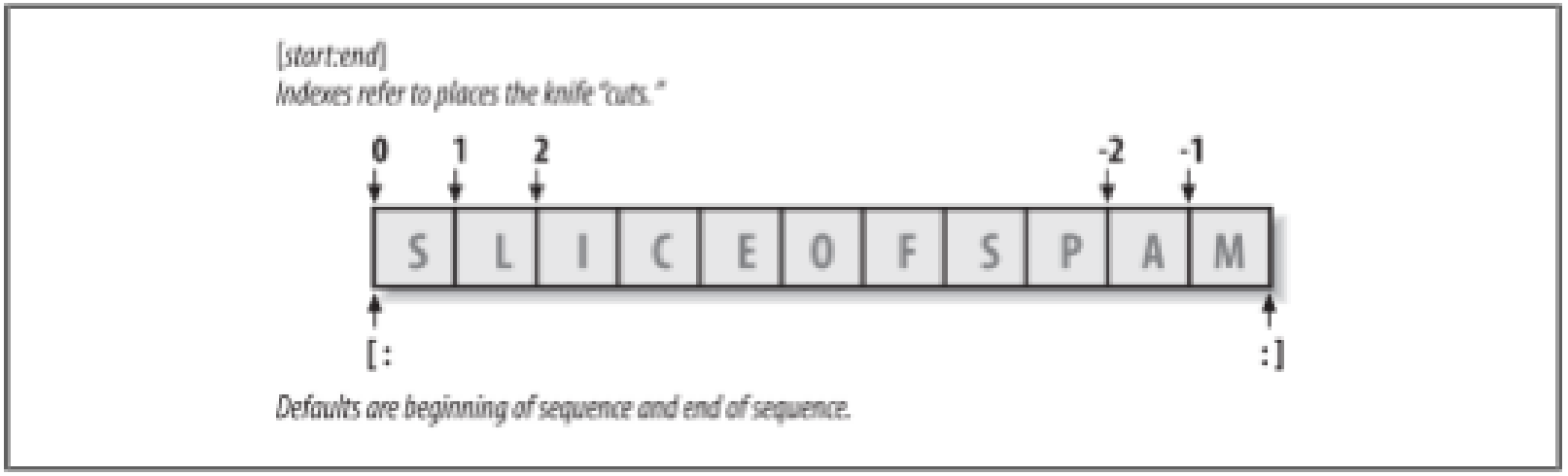
Loops through each letter in a string and counts the number of times the loop encounters the 'a' character

Example

```
word = 'Btechallbranches'
count = 0
for letter in word :
    if letter == 'a' :
        count = count + 1
print (count)
```

Indexing and Slicing

- >>> S = 'spam'
- Last character in the string has index -1 and the one before it has index -2 and so on



String Operations - Slicing

- We can look at the section of a string using a colon operator
- The second number is one beyond the end of the slice - “up to but not including”
- If the second number is beyond the end of the string, it stops at the end
- If we leave off the first number or the last number of the slice, it is assumed to be the beginning or end of the string respectively

Properties of Slicing

- S[1:3] fetches items at offsets 1 up to but not including 3.
- S[1:] fetches items at offset 1 through the end (the sequence length).
- S[:3] fetches items at offset 0 up to but not including 3.
- S[:−1] fetches items at offset 0 up to but not including the last item.
- S[:] fetches items at offsets 0 through the end—making a top-level copy of S
- Extended slicing (S[i:j:k]) accepts a step (or stride) k, which defaults to +1:
- Allows for skipping items and reversing order

String Operations - Slicing

- *Example 1*
- *>>> s = 'Btech All'*
- *>>> print s[0:4]*
- *Btec*
- *>>> print s[6:7]*
- *A*
- *>>> print s[6:20]*
- *All*

- *Example 2*
- *>>> s = 'Btech Electronics'*
- *>>> print s[:2]*
- *Bt*
- *>>> print s[7:]*
- *Lectronics*
- *>>> print s[:]*
- *Btech Electronics*

Extended slicing Examples

- >>> S = 'abcdefghijklmnop'
- >>> S[1:10:2] # Skipping items
- 'bdfhj'
- >>> S[::2] 'acegikmo‘
- >>> S = 'hello'
- >>> S[::−1] # Reversing items
- 'olleh'

str(42)
int(“42”)
repr(42)
ord(‘a’)
chr(97)

String Conversion Tools

- >>> "42" + 1
- TypeError: Can't convert 'int' object to str implicitly
- >>> int("42"), str(42)
- # Convert from/to string (42, '42')
- >>> repr(42)
- # Convert to as-code string '42'

Character code Conversions

- ord () - Convert a single character to its underlying integer code (e.g., its ASCII byte value)—this returns the actual binary value used to represent the corresponding character in memory.

- chr () - performs the inverse operation, taking an integer code and converting it to corresponding character

Character code Conversions - Example

```
>>> ord('s')
115
>>> chr(115)
's'
>>> S = '5'
>>> S = chr(ord(S) + 1)
>>> S
'6‘
 >>> S = chr(ord(S) + 1)
>>> S
'7'
```

Character code Conversions - Example

```
>>> int('5')
5
>>> ord('5') - ord('0')
5
>>> int('1101', 2)
 # Convert binary to integer: built-in
13
>>> bin(13)
# Convert integer to binary: built-in
'0b1101'
```

Changing Strings

- “immutable sequence”
- Immutable part means that you cannot change a string in place—for instance, by assigning to an index:
- >>> S = 'spam' # works
- #new piece of memory is allocated and named as S
- >>> S[0] = 'x' # Raises an error!
- TypeError: 'str' object does not support item assignment

Changing Strings

```
>>> S = S + 'SPAM!'

# To change a string, make a new one

>>> S

'spamSPAM!'

>>>S = S[:4] + 'Burger' + S[−1]

>>> S

'spamBurger!‘
```

Changing Strings

```
>>> S = 'splot'

>>> S = S.replace('pl', 'pamal')

>>> S

'spamalot'
```

Formating Strings

```
>>> 'That is %d %s bird!' % (1, 'dead')

# Format expression

That is 1 dead bird!

>>> 'That is {0} {1} bird!'.format(1, 'dead')

# Format method in 2.6, 2.7, 3.X

'That is 1 dead bird!'
```

Lower and upper

- Python has a number of string functions which are in the string library

```
>>> greet = 'Hello Arun'
>>> zap = greet.lower()
>>> print (zap)
hello arun
>>> print (greet)
'Hello Arun'
>>>zap1 = greet.upper()
>>> print(zap1)
HELLO ARUN
```

rstrip

```
>>>str = '    abc  '
>>> rstr=str.rstrip()
>>> str
'    abc  '
>>> rstr
'    abc’
```

Has an optional character argument to remove for example in the following code it removes ‘*’ from right side

```
>>>str = '    abc  **'
>>> rstr=str.rstrip('*')
>>> rstr
'    abc  ‘
```

String Capitalize

```
>>>str = 'hello'
>>> s1 = str.capitalize()
# make first letter of string as capital
>>> print(s1)
Hello
```

Centre

returns centered in a string of length width. Padding is done using the specified fillchar. Default filler is a space.

Syntax

str.center(width,[fillchar])

width - total width of the string

fillchar - filler character

```
>>>s1 = str.center(50)
>>> s1
'                    abc                    ‘
>>>s1 = str.center(50,'*')
>>> s1
'**********************  abc  **********************'
```

Searching a String

- finds the first occurrence of the substring
- If the substring is not found, find() returns -1

```
>>> name = 'pradeepkumar'
>>> pos = name.find('de')
>>> print (pos)
3
>>> aa = name.find('z')
>>> print (aa)
-1
```

Search and replace

- replace() - replaces all occurrences of the search string with the replacement string

```
>>> greet = 'Hello Kumar'
>>> nstr = greet.replace('Kumar','John')
>>> print (nstr)
Hello John
>>> nstr = greet.replace('e','O')
>>> print (nstr)
Hollo Kumar
```

Other Common String Methods in Action

- >>> line = "The knights who say Ni!\n“
- >>> line.rstrip()
- 'The knights who say Ni!'
- >>> line.upper()
- 'THE KNIGHTS WHO SAY NI!\n'
- >>> line.isalpha()
- False
- >>> line.endswith('Ni!\n')
- True
- >>> line.startswith('The')
- True

Other Common String Methods in Action

- length and slicing operations can be used to mimic endswith:
- >>> line 'The knights who say Ni!\n'
- >>> line.find('Ni') != −1
- # Search via method call or expression
- True
- >>> 'Ni' in line
- True
- >>> sub = 'Ni!\n'
- >>> line.endswith(sub)
- # End test via method call or slice True
- >>> line[-len(sub):] == sub
- True

Problem

In any of the country's official documents, the PAN number is listed as follows

<char><char><char><char><char><digit><digit><digit><digit><char>

Your task is to figure out if the PAN number is valid or not. A valid PAN number will have all its letters in uppercase and digits in the same order as listed above.

Problem

```
pan=input()
if(len(pan)==10)and
(pan.isupper())and(pan[0:5].isalpha())and
(pan[5:9].isdigit()) and (pan[9].isalpha()):
    print(pan," -Valid PAN")
else:
    print(pan," - Invalid PAN")
```

Problem

Find the sum of ascii values of your name.

Problem

Find the sum of ascii values of your name.

```
str=input("Enter your name")
total=0
for x in str:
    print(ord(x))
    total+=ord(x)
print("Total = ",total)
```

Write a program to check whether the given word is a palindrome.

```
mystr = input("Enter a string: ")

mystr = mystr.lower()

revstr = mystr[::-1]
print(revstr)

if mystr == revstr:
   print("It is palindrome")
else:
   print("It is not palindrome")
```

Encrypt

- Develop a security system that works with the caeser cipher.
- Let the given string be “Python” and the key is 3 letters away from each letter of the alphabet in lower case.
- So “**python**” becomes **“sbwkrq”**
  - p -> s
  - y -> b
  - t -> w
  - h-> k
  - o-> r
  - n-> q

```
cipher=
for ch in plaintext:
    n=ord(ch)+key
    if(n>ord('z')):
        n=n-26
    cipher=cipher+chr(n)
print("The Ciphertext is "+cipher)
```

Exercise

1. Write a program to count the number of vowels in your name.
2. Write a program to check whether the given two words are anagrams.