BCSE101E

Computer Programming: Python

Course Faculty: Rajesh M – SCOPE, VIT Chennai

Module - 5
Strings and Regular Expressions

Strings in Python

Check Validity of a PAN

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

<alphabet>< alphabet> < alphabet > < alphabet >

< alphabet > <digit><digit><digit><digit>< alphabet >

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.

PAC For PAN Problem

Input	Processing	Output
PAN number	Take each character and check if alphabets and digits are appropriately placed	Print Valid or Invalid

Pseudocode

```
READ PAN
If length of PAN is not ten then print "Invalid" and exit
FOR x=0 to5
        if PAN[x] is not a character THEN
                PRINT 'invalid'
                BREAK;
        END IF
END FOR
FOR x=5 to 9
        if PAN[x] is not a digit THEN
                PRINT 'invalid'
                BREAK;
        END IF
END FOR
IF PAN[9] is not a character THEN
        PRINT 'invalid'
                END IF
PRINT 'valid'
```

Test Case 1

abcde1234r

Valid

Test Case 2

abcde12345

Invalid

Test Case 3

abcd01234r

Invalid

Strings

- Immutable sequence of characters
- A string literal uses quotes
- 'Hello' or "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)
`B = b'sp\xc4m'`	Byte strings in 2.6, 2.7, and 3.X (Chapter 4, Chapter 37)
`U = u'sp\u00c4m'`	Unicode strings in 2.X and 3.3+ (Chapter 4, Chapter 37)
`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,

Example Strings

- 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)

Length of a String

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

>>> s

'a\x00b\x00c'

>>> len(s)

5

>>> print(s)

a b c
```

Length of a String

- —a binary 1 and 2 (coded in octal), followed by a binary 3 (coded in hexadecimal):

```
>>> s = '\001\002\x03'
>>> s
'\x01\x02\x03'
>>> len(s)
3
```

Backslash in Strings

- 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
```

Backslash in Strings

```
>>> x = "C:\py\code"

>>> x

'C:\\py\\code'

>>> len(x)

10
```

Check this

```
s = "C:\new\text.dat"

>>>s

print(s)

s1 = r"C:\new\text.dat"

>>>s1

print(s1)

s2 = "C:\\new\\text.dat"

print(s2)

>>>s2
```

Opening a File

- myfile = open('C:\new\text.dat', 'w') - Error
- 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'
- >>> print(path) # User-friendly format C:\new\text.dat
- >>> len(path)
- 15

Basic Operations

```
>>> 'Ni!' * 4
'Ni!Ni!Ni!Ni!'
>>> print('-' * 80)              # 80 dashes, the easy way
>>> myjob = "hacker"
>>> for c in myjob:
        print(c, end=' ')
h a c k e r
```

Basic Operations

```
>>> for c in myjob:
        print(c, end=' ')
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

Counting

- Count the number of 'a'

Counting

- Count the number of 'a'

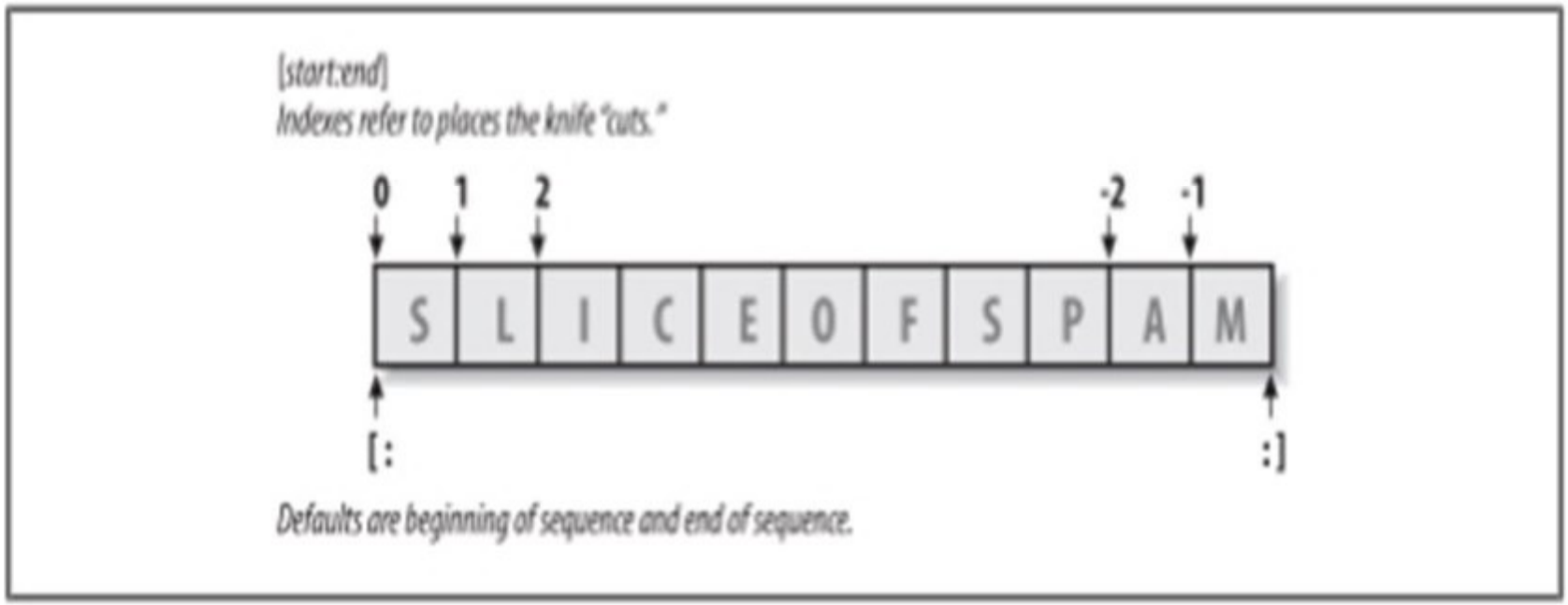
Example

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

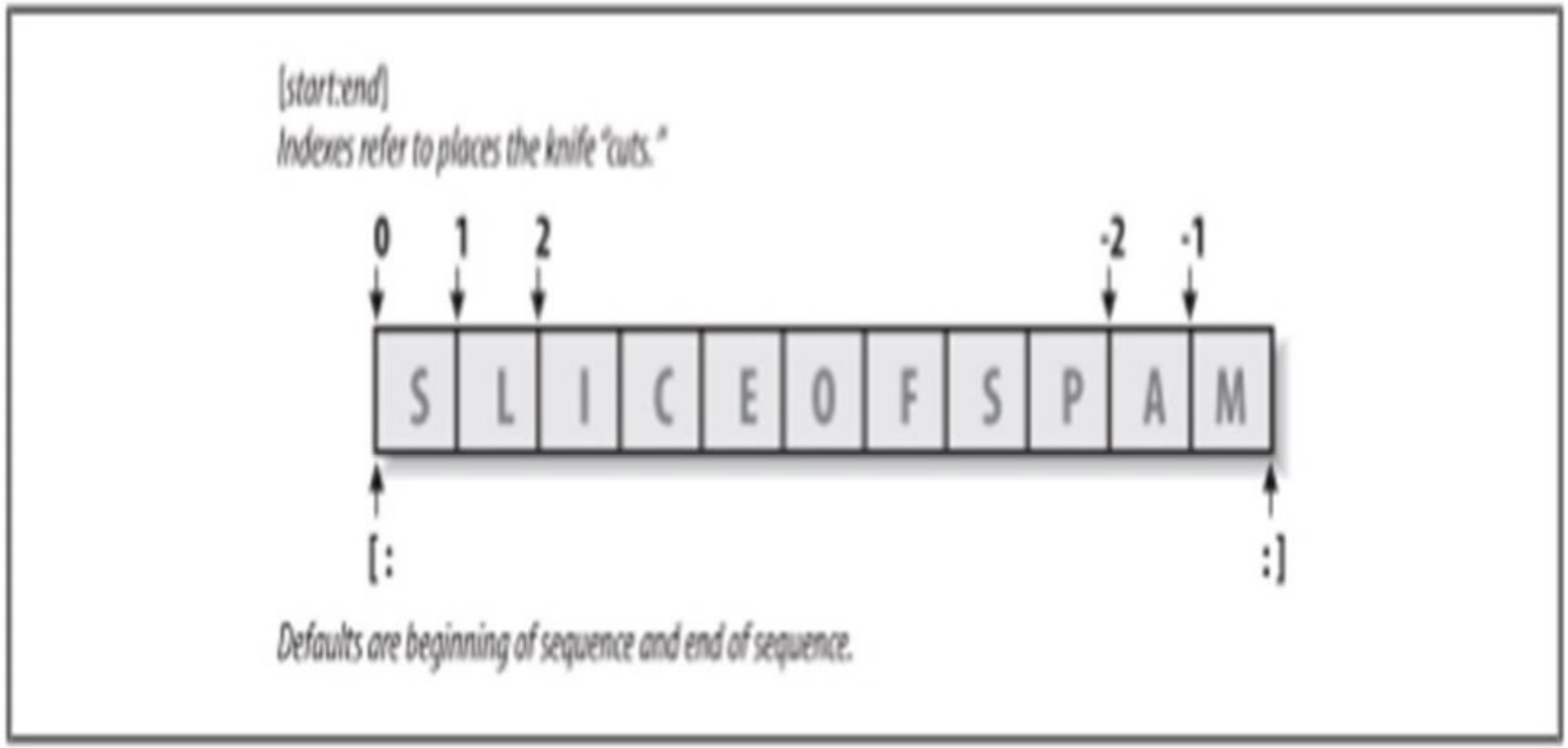
Indexing and Slicing

>>> S = 'spam'

Last character in the string has index -1 and the one before it has index -2 and so on



Indexing and Slicing



Indexing and Slicing

- Take one letter from a word at a time
- Use square bracket and give the index of the letter to be extracted
- Indexing can be done either from front or from end
- >>> S[0], S[-2]
- ('s', 'a')

Slicing

- Take a part of a word
- Square bracket with two arguments with a colon
- First value indicates the starting position of the slice and second value indicates the stop position of the slice
- Character at the stop position is not included in the slice
- >>> **S[1:3]**
- 'pa'

Slicing

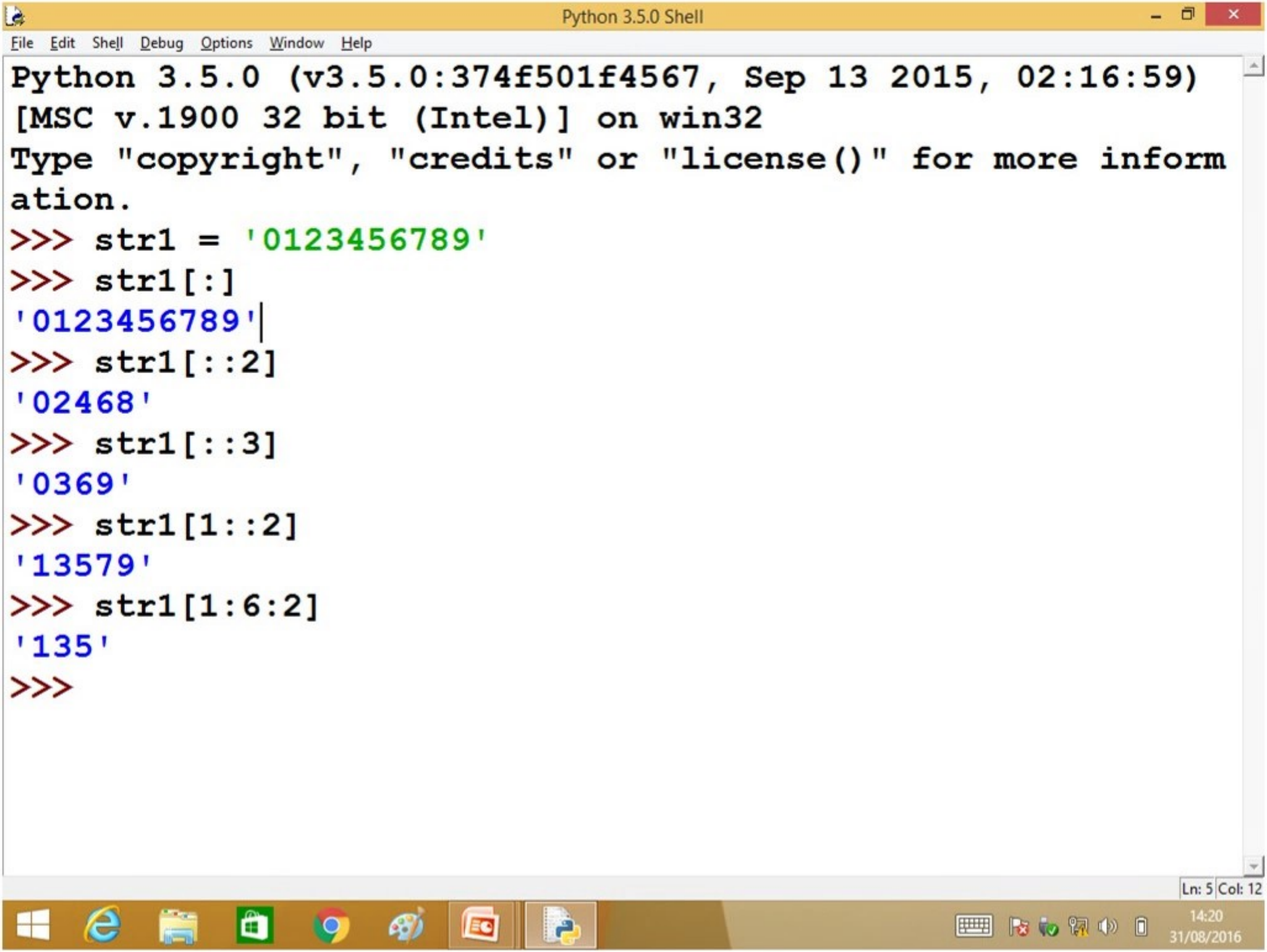
- If the second number is beyond the end of the string, it stops at the end
- If we leave off the first or last number of the slice, it is assumed to be beginning or end of the string respectively
- s = 'spam'
- >>>s[:3]
- 'spa'
- >>>s[1:]
- 'pam'

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**
- 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 last item**
- S[:] - fetches items at offsets **0 through the end** —making a top-level copy of S

Extended slicing

- **X[I:J:K]** - means "extract all the items in X, from offset **I through J–1, by K**."
- Third limit, K, **defaults to +1**
- If you specify an explicit value it is used to skip items
- Extraction is reversed when negative value is given for K-1
- Each time K-1 items are skipped


Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:16:59)
[MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more inform
ation.
>>> str1 = '0123456789'
>>> str1[:]
'0123456789'
>>> str1[::2]
'02468'
>>> str1[::3]
'0369'
>>> str1[1::2]
'13579'
>>> str1[1:6:2]
'135'
>>>
Ln: 5 Col: 12
14:20
31/08/2016

Extended slicing Examples

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

String Conversion Tools

- >>> **"42" + 1**
- **TypeError:** Can't convert 'int' object to str implicitly
- >>> **int("42"), str(42)** # Convert from/to string
- (42, '42')
- int("42") + 1
- 43
- >>> "42" + str(1)
- '421'

Character code Conversions

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

- >>> **ord('s')**

  115

Character code Conversions

chr () – Does **inverse of ord**

```
>>> chr(115)
's'
```

Character code Conversions - Example

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

Character code Conversions - Example

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

Concatenation

```
>>>S1 = 'Welcome'
>>>S2 = 'Python'
>>>S3 = S1 + S2
>>>S3
'WelcomePython'
```

Changing Strings

String - "**immutable sequence**"

Immutable - you cannot change a string in place

```
>>> S = 'spam'
>>> S[0] = 'x'                # Raises an error!
```

TypeError: 'str' object does not support item assignment

But S = 'Apple' works

How??

Changing Strings

```
>>> S = S + 'SPAM!'
# To change a string, make a new one
>>> S
'spamSPAM!'
>>> S = S[:4] + 'Burger' + S[-1]
>>> S
'spamBurger!'
```

Replace

```
>>> S = 'splot'
>>> S = S.replace('pl', 'pamal')
>>> S
'spamalot'
```

Formatting Strings

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

That is 1 dead bird!

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

'That is 1 dead bird!'

String Library

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

- These functions **do not modify the original string**, instead they return a new string that has been altered.

String Library

```
>>> greet = 'Hello Arun'
>>> zap = greet.lower()
>>> print (zap)
      hello arun


>>> print ('Hi There'.lower())
          hi there
```

Searching a String

find() - function to **search for a string** within another

find() - finds the **first occurrence of the substring**

If the substring is **not found**, find() **returns -1**

```
>>> name = 'kumar'
>>> pos = name.find('ma')
>>> print (pos)
2
```

Searching a String

```
>>> aa = "fruit".find('z')

>>> print (aa)

-1



>>> name = 'pradeepkumar'

>>> pos = name.find('de',5,8)

>>>pos

-1
```

Other Common String Methods in Action

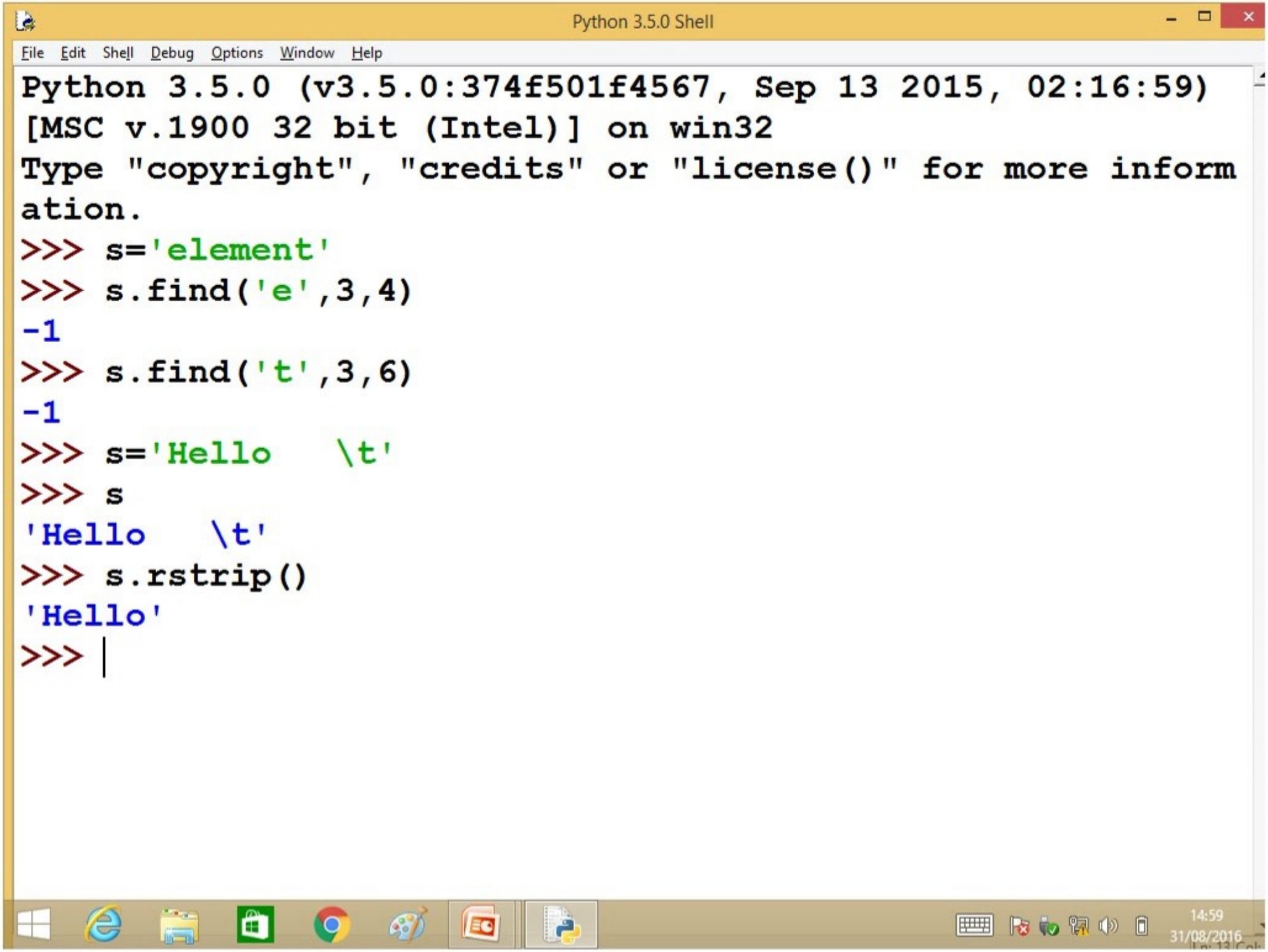
```
>>> line = "The knights who say Ni!\n"

>>> line.rstrip()

'TheknightswhosayNi!'

>>> line.upper()

'THE KNIGHTS WHO SAY NI!'
```


Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:16:59)
[MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more inform
ation.
>>> s='element'
>>> s.find('e',3,4)
-1
>>> s.find('t',3,6)
-1
>>> s='Hello \t'
>>> s
'Hello \t'
>>> s.rstrip()
'Hello'
>>>
14:59
31/08/2016

Other Common String Methods in Action

- >>> 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
True
>>> 'Ni' in line
True
```

Other Common String Methods in Action

```
>>> sub = 'Ni!\n'
>>> line.endswith(sub)  # End test via method call or slice
True
>>> line[-len(sub):] == sub
True
```

To check if all letters in a String are in Uppercase

isupper() function is used to check if all letters in a string are in upper case.

Examples

```
>>> 'a'.isupper()
False
>>> 'A'.isupper()
True
>>> 'AB'.isupper()
True
>>> 'ABc'.isupper()
False
>>>
```

To check if all letters in a String are in Lowercase

islower() function is used to check if all letters in a string are in lower case.

Examples

```
>>> 'a'.islower()
True
>>> 'aB'.islower()
False
```

To check if a sentence is in Title case

istitle() function is used to check if all letters in a string are in upper case.

Examples

```
>>> 'Apple Is A Tree'.istitle()
True


>>> 'Apple Is A tree'.istitle()
False
```

```
pan = input("enter pan")
invalid = False
if len(pan)!=10:
    invalid = True
else:
    for i in range(0,5):
        if not pan[i].isalpha():
            invalid = True
            break
    for i in range(5,9):
        if not pan[i].isdigit():
            invalid = True
            break
if not pan[9].isalpha():
            invalid = True
if invalid == True:
    print ("Invalid")
```