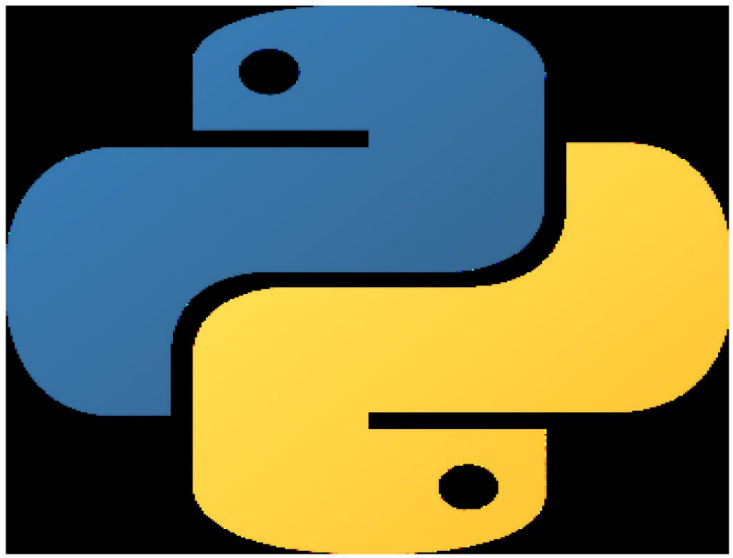


Programming with Python

2. Strings

What is a String?

- A string is **a sequence of characters**
- A string literal uses quotes – Single or Double

  Ex: 'Hello' or "Hello"
- For strings, + means "concatenate"
- When a string contains numbers, it is still a string
- We can convert numbers string into a number using int( )

```
>>> str1 = "ACTS"
>>> str2 = „DBDA'
>>>  str3 = str1 + str2
>>> print
str3 ACTSDBDA
```

```
>>> str4 = '123'
>>> str5 = str4 + 1
Traceback (most recent call
last): File "<stdin>", line 1,
in <module>
TypeError: cannot concatenate
'str' and 'int' objects
```

```
>>> x = int(str3) + 1
>>> print x
124
```

Multiline strings

- String literals can be defined with single quotes or double quotes.
- Can use other type of quotes inside the string.

```
>>> str = "I am „Bond"..."James Bond""
"I am 'Bond'...'James Bond'"
```

- Multiline strings can be initialized using “““str,,””” or ” ” ” str “ ” ”

```
>>> str = """ I am
                "Bond"..."James Bond"
            """
>>> str
'I am \n"Bond"..."James Bond" \n'
```

String as input

- We can read data as strings and then parse and convert the data as we need.
- **raw_input** is **deprecated** in Python3

```
>>> name = input('Enter Name:')
Enter Name:Bond
>>> print(name)
Bond
```

```
>>> ID = input('Enter ID:')
Enter ID:100
>>> x = ID - 10
Traceback (most recent call last): File "<stdin>", line 1, in <module>TypeError: unsupported operand type(s) for -: 'str' and 'int'
>>> x = int(ID) - 10
>>> print(x)
90
```

Escape Characters

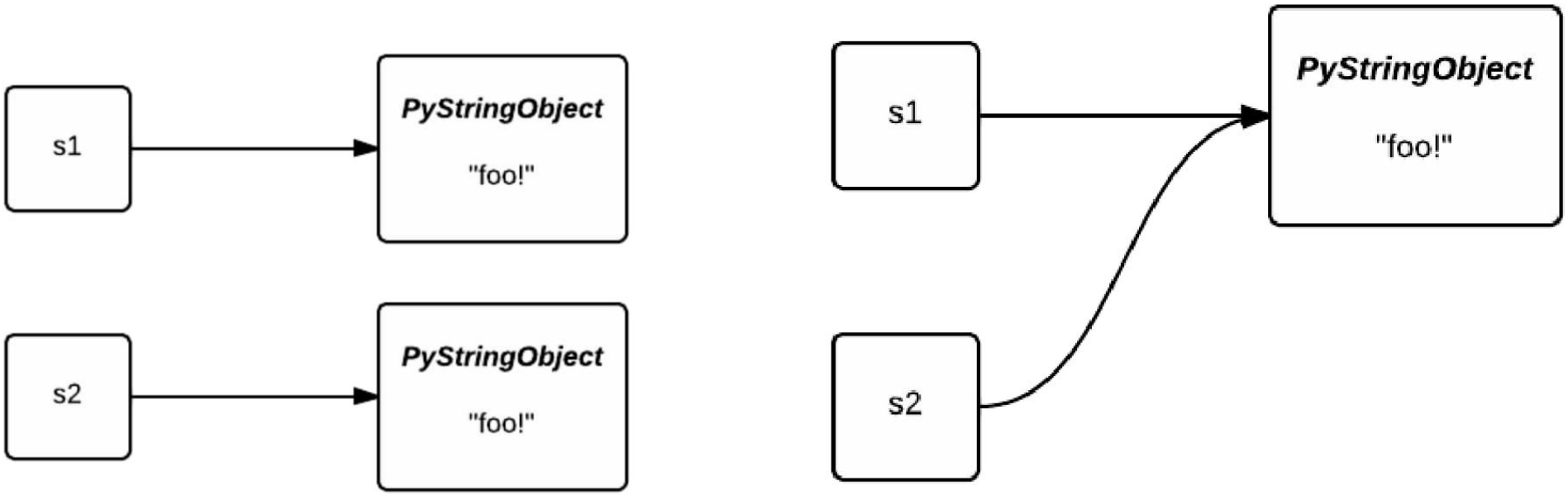
- Some characters need to be escaped in a string.

```
# Using single quotes inside a string
>>> print("My name is "Ram"")
SyntaxError: invalid  syntax
>>> print("My name is \"Ram\"")
Ram
```

```
# Line feed or New line char
>>> print("My name is \n Ram")
My name
 is Ram
```

Interning Strings

Firstly, “sharing” or Interning string objects reduces the amount of memory used.



Native Interning

Under certain conditions, strings are natively interned.

```
>>> s1 = 'foo'

>>> s2 = 'foo'

>>> s1 is s2

True
```

How to store strings?

"String is a collection of characters"

A **character** can be represented using its ASCII Values. Agree?

How to represent **"Snow Man"** char using ASCII?

ASCII is limited to 8 bits – 256 characters only, **can encode only Latin alphabet.**

ISO-8859-1	0x00 to 0x7F (ASCII)	!"#$%&\'()*+,-./0123456789:;<=>?@ ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_` abcdefghijklmnopqrstuvwxyz{|}~
	0x80 to 0xFF	¡¢£¤¥¦§¨©ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß àáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

How to store strings?...

ISO-8859-* Family

Each language requires a different set of character set.

- Latin/Arabic
- Latin/Greek
- Latin/Hebrew
- Western European
- Central European
- South European
- North European
- Latin/Cyrillic

Ref: PyCon3/2009-05-09 Unicode and Python 3

How to store strings?

How to represent **"Snow Man"** char?

Unicode Transformation Format – UTF-8 (Used by Python3)

U+2603

(Unicode hex value of snowman char in the category of "other symbols")

```
>>> snowman='\u2603„
>>> snowman
'☃'
```

What is Unicode?

- Universal character encoding standard used for representation of text for computer processing
- Provides a consistent way of encoding multilingual plain text
- Unicode Standard **assigns** each character a **unique numeric value (Code point), name, category, and some other attributes**
- Supports three encoding forms - **UTF-8**, UTF-16, UTF-32 Example:

A LATIN CAPITAL LETTER A U+0041

Char A — Name — Code Point (0 to 10FFFF)

0x0041 = 65 → Which is equal to the ASCII value of A.

Ref: http://unicode.org/standard/principles.html

Other way of storing strings – Byte Strings

- Python also supports Byte Strings
- String is represented as a sequence of bytes

Example:

```
>>> name="John"

>>> type(name)
<class 'str'>

>>> name=b"John" # Prefix b means it is a byte string
<class 'bytes'>
```

String types

- Unicode Strings
- Byte Strings

	Sequence of	Python 2	→	Python 3
Unicode strings	Codepoints	**u**"Unicode string"	→	"Unicode string"
		<type '**unicode**'>	→	<class '**str**'>
Byte strings	Bytes	"Byte string"	→	**b**"Byte string"
		<type '**str**'>	→	<class '**bytes**'>

Default:
Byte String

Default:
Unicode String

- Python 3.0, all strings are stored as Unicode in an instance of the str type

Encoding and Decoding Strings

- **Encoding:** **str type → bytes type**

  Unicode String → Byte String

  bytestr = unistr.encode('*utf-8*')

  Unicode String → ASCII

  bytestr = unistr.encode(„ascii')

```
>>> snowman = 'Snowman: ☃'
>>> bytestr = snowman.encode("utf-8")
b'Snowman: \xe2\x98\x83„

>>> bytestr.decode("utf-8")
'Snowman: ☃'
```

Encoded strings on the other hand are represented as binary data in the form of instances of the bytes type. **str – text, bytes - data**

Encoding and Decoding Strings

- **Decoding:** **bytes type → str type**

  Bytes → Unicode

  uni_string = byte_string.encode('*utf-8*')

  Bytes → ASCII

  ascii_string = byte_string.encode(„ascii')

```
>>> snowman = b'Snowman: ☃'
>>> snowman.encode('ascii')
UnicodeEncodeError: 'ascii' codec can't encode character '\u2603' in position 9: ordinal not in range(128)

>>> snowman.encode(„utf-8')
b'Snowman: \xe2\x98\x83'
```

Special Operators on Strings

Operator	Description	Example
+	Concatenation	“hello+world” = helloworld
*	Repetition	Hello*2 → HelloHello
[]	Access a char using index.	str=“hello” str[0] = „h“, str[1]=„e“, str[-1]=„o“ str[7] - IndexError: string index out of range
[:]	Slicing a string	str[0:3]=„hel“
in, not in	Membership	„h“ in str →True
r/R	Raw string – suppress the meaning of escape chars	print(r"'\n' Hello \n World") O/P: '\n' Hello \n World
%	String Formatting (Old Style) (Similar to C)	x = 12.3456789 print('The value of x is %3.2f' %x) O/P: 20.43

Modifying & Deleting a String

- Strings are **immutable**
- We can reassign completely a different string, cant modify a char.

Examples:

```
>>> name="John„
>>> name[0]="K"
TypeError: 'str' object does not support item assignment
>>> name="Kohn"
```

```
//Deleting a string
>>> name="John"
>>> del name
>>> type(name)
NameError: name 'name' is not defined
```

Accessing chars

- Can access a single character in a string using an index
- The index value must be an integer and starts at zero
- The index value can be an expression


```
>>> course="PGDBDA"
>>> print(course[2])
D
>>> i = 4
>>> print(course[i+1])
A
```

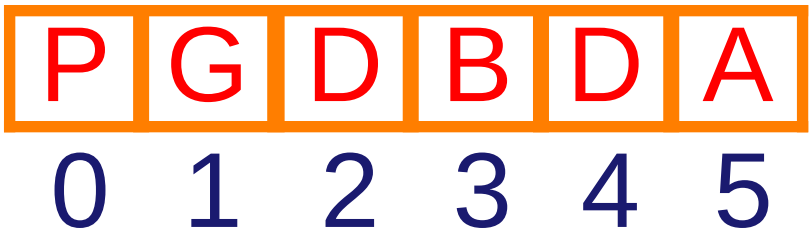
Length of a string

- Built-in function – len
- Gives length of a string
- No end of string character


```
>>> course="PGDBDA"
>>> print len(course)
6
```

Bounds Checking

- Python do strict bounds checking
- Run time error, if you go out of bounds




```
>>> course="PGDBDA"
>>> print course[6]
IndexError: string index out of range
```

Iterating through strings

- We can traverse through a string using a loop to look at each of the letters in a string individually
- Use a while statement and an iteration variable, and the len function.


```
>>> course=„PGDBDA„
>>> index = 0

>>> while index <
      len(course): ch =
      course[index] print(
      index, ch ) index =
      index + 1
```

Iterating through strings – Another way

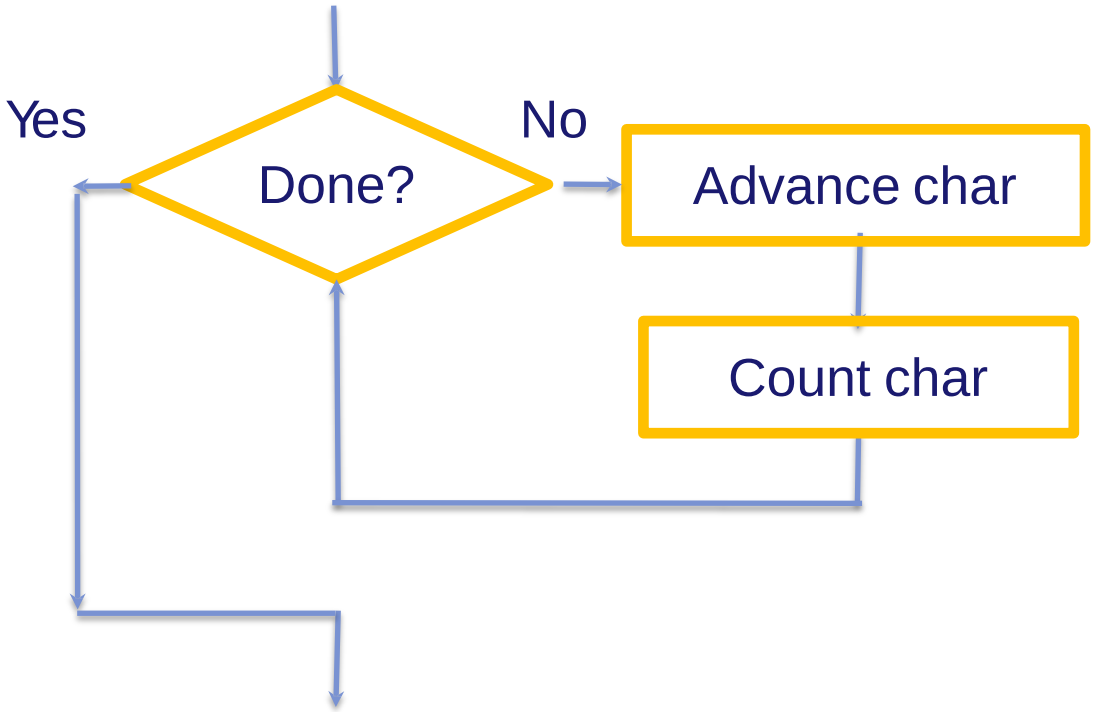
- Use a **for statement** and an **iteration variable**.
- More elegant than while
- Iteration/counter variable will be taken care by for.


```
>>> course=„PGDBDA„

>>> for ch in course:
      print(ch)
```

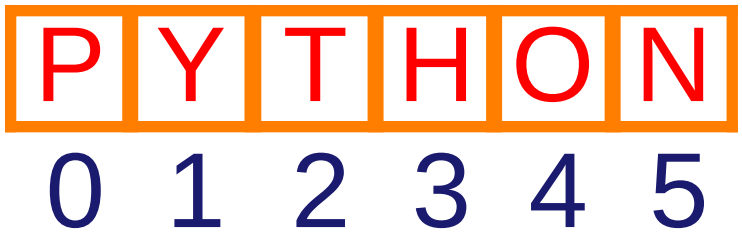
Find the length of a string

- Write a program to find out the length of a string using for loop and while loop.



Slicing Strings

- Used to extract a substring/subsection of a string
- Use colon (:) operator
- Name[m:n] → Gives chars from index "m" to index "n-1"
- If the second index is more than the length, then it will stop at the end

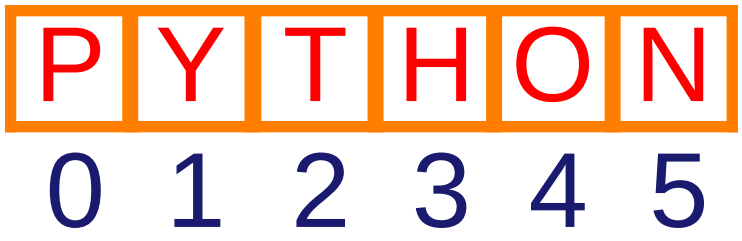



```
>>> course=„PYTHON„

>>> course[1:4]
YTH
```

Slicing Strings…

- Leaving out the last index → Means till the end
- Leaving out the first index → Means from the beginning




```
>>> course=„PYTHON„

>>>
course[:4]
PYTH

>>> course[3:]
HON
```

String Library

- Python has several built-in string library functions

```
>>> str1 = "Learning"
>>> dir(str1)
['__add__',    '_class__',      ' contains  ',      ' delattr  ',
'__dir__',    '__doc__',    '_eq__',     ' format ',  '  '__ge__',
'_ getattribute_ ',  ' getitem  ',   getnewargs  ',      '__gt__',
'__hash  ',    ' init  ',     ' iter  ',     '_le__',     '__len__',
'__lt__',     '__mod__',     '_mul__',      '_ne__',      '__new__',
'__reduce ',       ' reduce_ex  ',       ' repr  ',       '_ rmod  ',
' rmul  ',        ' setattr  ',        ' sizeof  ',        '__str__',
' subclasshook ',    'capitalize', 'casefold', 'center', 'count',
'encode',    'endswith',    'expandtabs',    'find',    'format',
'format_map',   'index',   'isalnum',   'isalpha',   'isdecimal',
'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace',   'istitle',   'isupper',   'join',   'ljust',   'lower',
'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

dir - Returns the attributes of the object or module.

String Library…help pages - Demo

>>> help(str)

```
ramesh@ramesh-ThinkCentre-M92p: ~
File  Edit  View  Search  Terminal  Help
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __format__(...)
 |      S.__format__(format_spec) -> str
 |
 |      Return a formatted version of S as described by format_spec.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
```

Searching in a string - find

- Search for a substring within another string – find()
- Syntax: S.find(sub[, start[, end]])
- find() finds the first occurrence of the substring
- If the substring is not found, returns -1

H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

```
>>> phrase = "Hello World"
>>> phrase.find("World")
6
>>> phrase.find("o", 3, 7)
4
>>> phrase.find('o', 5, 9)
7
>>> phrase.find('o', 0, 3)
-1
```

Find and Replace

- Search for a substring and replace with another substring – replace()
- Syntax: **S.replace(old, new[, count])**
- Replace all occurrences of the substring
- Return the string after the replacement

H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

```
>>> string = „Hello World'
>>> string.replace('World', 'India')
"Hello India"
>>> string
"Hello India" → Original string will not be overwritten
>>> string.replace('l', 'L',2)
'HeLLo World'
>>> string.replace(„m', 'L')
'Hello World„  → m is not present, did nothing
```

Removing Whitespaces

- lstrip() - Remove whitespaces at the beginning → left side
- rstrip() - Remove whitespaces at the end → Right side
- strip() - Remove whitespaces at the beginning and at the end

		H	e	l	l	o		W	o	r	l	d	
0	1	2	3	4	5	6	7	8	9	10	11	12	13

```
>>> string = „  Hello World „
>>> string.lstrip()
"Hello World "

>>> string.rstrip()
"  Hello World"

>>> string.strip()
"Hello World"
```

String Concatenation

- **1st Way:** Use + operator

```
>>> str1 = "Learning"
>>> str2 = " "
>>> str3 = "PYTHON"

>>> str = str1 + str2 + str3
>>> str
Learning Python
```

String Concatenation…

- **2nd Way:** Use method **"join" (Best Way)**

```
>>> str1 = " "    → Separator – Can be space or _ or -
>>> str2 ='I', 'am', 'learning', 'Python'
(OR)
>>> str2 = (I', 'am', 'learning', 'Python')

>>> str1.join(str2)
I am learning Python
```

- **3rd Way:** Use method **"format"**

```
>>> s="{0} {1} {2} {3}"  → Positional parameters
>>> s.format('I', 'am', 'learning', „Python')
I am learning Python
```

Example1

Check whether a string is palindrome or not

```
# take input from the user
my_str = input("Enter a string: ")

# reverse the string
rev_str = ''.join(reversed(my_str))

# check if the string is equal to its reverse
if my_str == rev_str:
   print("It is palindrome")
else:
   print("It is not palindrome")
```

Example2

Sort the words of a multiword string

```
# Program to sort alphabetically the words form a string provided by the user

# take input from the user
str1 = input("Enter a string: ")

# breakdown the string into a list of words
words = str1.split()

# sort the list
words.sort()

# display the sorted words
for word in words:
    print(word)
```

Exercises

1. Write a simple python program to reverse a string without using built-in functions.
2. Write a python program that takes a string and finds frequency listing of the characters contained in it.
3. A pangram is a sentence that contains all the letters of the English alphabet at least once, for example: The quick brown fox jumps over the lazy dog. Your task here is to write a function to check a sentence to see if it is a pangram or not.