Introduction to Programming Language (ITP101)

Unit 6: String Operations

Mulualem Teku

GCIT, Bhutan

Oct, 2019

...So Far in Python & Today...

- Core Python objects:
  - Functions
  - Lists
  - Tuples
  - Dictionaries
  - Sets
- Exception handling and debugging
  - `try...except...[else]`
  - `try...finally`
  - `assert`
  - The `pdb` debugger

Today:

- Strings

Brainstorm

1. Mutable vs Immutable data type? Give examples.

2. What are sequence data types (a.k.a. Sequences)? Examples?

3. What are strings? Some applications/use cases?

Brainstorm

1. Mutable vs Immutable data type? Give examples.

2. What are sequence data types (a.k.a. Sequences)? Examples?

3. What are strings? Some applications/use cases?

Brainstorm

1. Mutable vs Immutable data type? Give examples.
2. What are sequence data types (a.k.a. Sequences)? Examples?
3. What are strings? Some applications/use cases?

- Everything in Python is an object.

Core Python Objects

- Numbers
- Boolean
- Functions
- Modules
- Lists
- Tuples
- Dictionaries
- Strings
- Files
- Classes

- Mutable vs Immutable objects

Recap: Sequence Data Types

- A.k.a. Sequences are positionally *ordered* set of objects.
- Notion of left-to-right ordering.

Some Built-in Objects

Lists ✓

Tuples ✓

```
Strings  ✓
Dictionaries              # unordered mapping type
Sets                      # unordered collection
```

- Mutable vs Immutable sequences

Strings

- Sequence of bytes or characters.

  e.g. gene sequence, database records, text files, binaries, etc

- No `character (char)` data type in Python.
- A character is just a string of length 1.
- String literals

Example

```
>>>str = ‘CCCAAGGTTTTTAGGCCCT’

>>>str = "To be, or not to be, that is the question"

>>>str = ‘‘‘this is also a string literal            # multi-line string
that spans multiple lines ’’’

>>>str = """ similar to the above triple quote """    # multi-line string

>>>print (str)

>>>print (type(str))
```

- Strings are immutable .

```
>>>name='''Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch'''

>>>name[0]

>>>name[-1]                        # negative indexing

>>>name[3]='e'                     # ??

>>>name[100]                       # ??
```

- The usual escape sequences apply

```
>>>print """Faith, hope and love remain. \n But the greatest

of these is \t love"""
```

String Operations

- Concatenation (+)

```
>>>‘ATG’ + ‘CAGAT’
```

- Repetetion (*)

```
>>>"Hello" * 10
```

- Indexing

```
str[index]
```

- Slicing

```
str[start] - str[end-1]
```

```
>>>S=‘AGGTTTCCCCCG’
```

```
>>>S[2:5]
>>>S[:4]
>>>S[6:]
>>>S[:]
>>>S[1:8:2]
```

- Membership checking

```
in / not in
```

String Methods

Assume `txt = "I love Bhutan."`

- `str()`
- `len(txt)`
- `isalpha()`, `isdigit()`, `isspace()`, etc
- `find('sub')`
- `replace('old','new')`
- `count('sub')`

String Methods (2)

- `split()`
- `strip()`
- `upper(), lower()`
- `join(sequence)` sequence = string, list, tuple ...
- `startswith('sub'), endswith('sub')`

String Formatting

- Python offers various string formating facility:

  i) Using the '%' operator. (old Python)

Format Specifiers

- `%d` (int)
- `%s` (string)
- `%o` (octal)
- `%x` (hex)
- `%f` (floating point)
- `%g` (floating point)

- Usage: `<format strings> %(<matching values>)`

Examples

```
from math import pi

name = input("What is your name?")

r = int(input("Radius: "))

print ( "Hi %s , your radius is %d and the area is %f." %(name, r, pi*r*r) )
```

ii) Using the format() method. (new Python)

- How can you combine/concatenate strings and numbers in Python?
- The **format()** method accepts arguments passed to it, formats them, and places them in a string containing curly brace placeholders i.e. $\{\}$.

ii) Using the format() method. (new Python)

- How can you combine/concatenate strings and numbers in Python?
- The **format()** method accepts arguments passed to it, formats them, and places them in a string containing curly brace placeholders i.e. {}.

Example

```
x = 39
y = "Bhutan"
wish = "His majesty the king of {} is {} years old. May he live long."

# placeholders can also be written as {0}, {1}, {2}, {3}, etc

print(wish.format(y, x))
```

Strings: Problem solving

Examples

1.
2.
3.
4.
5.