Programing in Python

Lecture 5 - Strings

Instructor: Zhandos Yessenbayev

Outline

- Python Strings
- Escape Characters
- Indexing and Slicing
- String Methods
- Parsing Strings
- Formating Strings

Python Strings

- **String** is a sequence of Unicode characters (alphabet, symbols)

- To represent a string wrap a text within *quotes*:

  - **Single** quotes - ' Hello, friends! '

  - **Double** quotes - " My name is Tom."

  - **Tripple double** quotes - """ From: info@example.com """

  - **Tripple sigle** quotes - ''' From: info@example.com ‘''

Python Strings

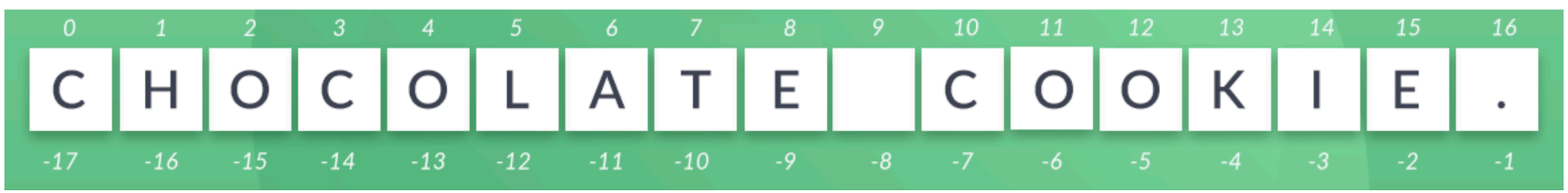
- *single_quote* = 'Single quote allow you to embed "double" quotes in your string.'
- *double_quote* = "Double quote allow you to embed 'single' quotes in your string."
- *triple_quote* = """Triple quotes allows to embed "double quotes" as well as 'single quotes' in your string. And can also span across multiple lines."""

Escape Characters

- Special **escape characters** can be used in string with “\”:
  - quoted_text = "I said \"Hello!\" to you.”
  - multiline_text = "First line\n Second line\n"
  - tabbed_text = "First line\n \t Tabbed second line\n"
  - backslashed_text = "C:\\Program Files\\"

Indexing and Slicing

- Strings are **immutable arrays**, i.e. we *can access* each character by index, but *cannot change* them




```
>>> choko = "CHOCOLATE COOKIE."
>>> len(choko)
17
```

len( ) — returns string length

Indexing and Slicing

```
>>> choko = "CHOCOLATE COOKIE."
>>>
>>> choko[0]
'C'
>>> choko[1]
‘H'
>>> choko[16]
'.'
>>> choko[-1]
'.'
>>> choko[-2]
'E'
>>> choko[0] = 'M'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Indexing and Slicing

- To take a **range** of characters, we use string **slicing**.

```
>>> new_str = "Hello, friends!”
>>>
>>> new_str[0:5]
'Hello'
>>> new_str[7:]
'friends!'
>>> new_str[:-1]
'Hello, friends'
>>> new_str[-5:-2]
‘end'
>>> new_str[:]
>>> new_str[::-1]
```

String comparison

- Strings can be compared using lexicographic order

```
>>> word = 'banana'
>>>
>>> if word < 'banana':
...     print('Your word,' + word + ', comes before banana.')
... elif word > 'banana':
...     print('Your word,' + word + ', comes after banana.')
... else:
...     print('All right, bananas.')
...
All right, bananas.
```

String Methods

- Find about string methods using **dir** function:

```
>>> dir(new_str)
['__add__', '__class__', '__contains__', '__delattr__',
'__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__', '__init__', '__le__',
'__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmod__', '__rmul__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', '_formatter_field_name_split',
'_formatter_parser', 'capitalize', 'center', 'count',
'decode', 'encode', 'endswith', 'expandtabs', 'find',
'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
```

String Methods

```
>>> S = "This is my book"
>>> S.count(‘i')
>>> 2
>>>
>>> S.startswith('This')
>>> True
>>>
>>> S.find(‘my’)
>>> 8
>>>
>>> S.isalnum()
>>> S.isalpha()
>>> S.isdigit()
>>> S.isspace()
>>> S.islower()
>>> S.isupper()
```

```
>>> S = "This is my book"
>>>
>>> S.lower()
>>> 'this is my book'
>>>
>>> S.upper()
>>> 'THIS IS MY BOOK'
>>>
>>> S.replace('my', ‘your')
>>> 'This is your book'
>>>
>>> S.split(' ')
>>> [‘This', ‘is', ‘my', 'book']
```

Parsing Strings

- Looping through characters:

```
>>> word = 'banana'
>>> count = 0
>>> for letter in word:
...     if letter == 'a':
...         count = count + 1
...
>>> print(count)
>>> 3
```

Parsing Strings

- Use string functions to parse useful information:

```
>>> data = 'From bob@gmail.com Sat Jan  5 09:14:16 2008'
>>> tokens = data.split(' ')  # split string by spaces
>>> for token in tokens:
...     if '@' in token:  # get email address
...        print("Email: " + token)
...     if ':' in token:  # get time
...        print("Time: " + token)
...
>>>
```

Formating Strings

- Format strings for pretty printing:
  - add strings with **“+”**
  - add helping symbols (**spaces, tabs, new lines, comma**)
  - convert non-string variables to string with **str( )**

```
>>> name = “Tom"
>>> last_name = "Johnson"
>>> age = 35
>>> S = "Client: \n\t” + name + " " + last_name + ", " + str(age)
>>> S
'Client: \n\tTom Johnson, 35'
>>>
>>> print(S)
Client:
        Tom Johnson, 35
```

Formating Strings

- We can use % function

2 placeholders

2 values

```
>>> print("I bought %s for %d Euros!" %('cookies', 200))
>>> I bought cookies for 200 Euros!
>>>
>>> camels = 42
>>> '%d' % camels
>>>
```

%d - integer
%s - string
%f - floating number

Formating Strings

- Also we can use **format( )** function

```
>>> name = “Tom"
>>> last_name = "Johnson"
>>> age = 35
>>> S = "Client: \n\t {} {}, {}"
>>> S
'Client: \n\t {} {}, {}'
>>>
>>> print( S.format(name, last_name, age) )
Client:
        Tom Johnson, 35
```

Formating Strings

- More formatting:

```
>>> # Refer by Index Numbers
>>> age = 36
>>> name = "John"
>>> txt = "His name is {1}. {1} is {0} years old."
>>> print(txt.format(age, name))
>>>
>>> # Refer by Names
>>> myorder = "I have a {carname}, it is a {model}."
>>> print(myorder.format(carname = "Ford", model = “Mustang"))
>>>
>>> txt = " Price is {:.2f} dollars.”
>>> print(txt.format(25))
```

Thanks!