Lecture 1c Basic Python : About Strings

A **string** is any sequence of characters. We usually use them to store words, sentences or even entire paragraphs.

The data type of a string is `str`.

Creating a String

String can begin and end with either single quotes or double quotes.

>>> spam = “That is Alice.”

or

>>> spam = ‘That is Alice.’

String can contain single quote and we can type :

>>> spam = “That is Alice’s cat.”

But not

>>> spam = ‘That is Alice’s cat.’ (This will not work !)

Escape Characters

Escape Characters	Prints as	Example	Output String
\’	Single quote	>>> ‘Alice\’s cat.’	Alice’s cat.
\”	Double quote	>>> “Type the \“Double Quote\” here.”	Type the “Double Quote” here.”
\t	Tab	>>> print(‘Class \tName \tTel’)	Class Name Tel
\n	Newline (line break)	>>> print(‘Class \nName \nTel’)	Class Name Tel
\\	Backslash	>>> ‘Type the backslash \\ here.’	Type the backslash \ here.

>>> spam = ‘That is Alice\’s cat.’

(This will work now!)

Comments

The hash character (#) marks the beginning of a comment within a Python code.

Eg .

```
# Comment for Python program.
```

Within the Python IDLE, we can use alt-3 to comment a line and alt-4 to un-comment the line.

Multi-line Comment

The triple quotes, either single or double, can be used to marks the beginning and end of a multi-line comment within a Python code.

Eg .

```
'''This is a test Python program.
Use this as a template.
Do not remove any line.'''
```

The *in* and *not in* Operators with Strings

The operators `in` and `not in` will evaluate to a Boolean `True` or `False`.

Eg .

```
>>> 'Hello' in 'Hello World'
True
>>> 'HELLO' in 'Hello World'
False
```

The + and * Operators with Strings

The arithmetic operators + and * can be use for string concatenation and replication.

Eg .

```
>>> 'Hello ' + 'World'
'Hello World'
>>> 'HELLO ' * 3
'HELLO HELLO HELLO '
```

The >, <, ==, >=, and <= Operators with Strings

The inequality operators can be use to compare strings they will evaluate to a Boolean `True` or `False`.

Eg .

```
>>> 'Ten' > 'One'
True
>>> 'FIVE' == 5
False
```

Some String Operations

```
>>> s = 'ba'
>>> t = 'ck'

>>> s + t
'back'

>>> t = s + 'na' * 2
>>> t
'banana'
```

```
>>> 'z' in t
False

>>> 'bananb' > t
True

>>> 'banan' <= t
True

>>> 'c' < t
False
```

Useful String Methods

```
>>> len('Hello World')
11
```

There are other useful methods but we will not cover them in our lesson.

Eg: `upper(), lower(), isupper(), islower(), isalpha(), isalnum(), isdecimal(), isspace(), istitle(), startswith(), endswith(), join(), split(), rjust(), ljst(), center(), strip(), rstrip(), lstrip()`

Index Operator on String (video)

Index a String

Example :

```
>>> s =  'abcd'
>>> s[0]
  'a'
>>> s[2]
  'c'
```

First character is indexed with `0`.

String Slicing (video)

String Slicing with Step (video)

String Slicing

`s[start:stop:step]`

non-inclusive

```
>>> s =  'abcdef'
>>> s[0:2]
 'ab'
>>> s[1:2]
 'b'
>>> s[:2]
 'ab'
```

```
>>> s[1:5:3]
 'be'

>>> s[::2]
 'ace'
```

Important : Slicing always returns a new string.

Operation	Result	Notes
`x in s`	`True` if an item of *s* is equal to *x*, else `False`	(1)
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`	(1)
`s + t`	the concatenation of *s* and *t*	(6)(7)
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times	(2)(7)
`s[i]`	*i*th item of *s*, origin 0	(3)
`s[i:j]`	slice of *s* from *i* to *j*	(3)(4)
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*	(3)(5)
`len(s)`	length of *s*	
`min(s)`	smallest item of *s*	
`max(s)`	largest item of *s*	
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)	(8)
`s.count(x)`	total number of occurrences of *x* in *s*	

3. If *i* or *j* is negative, the index is relative to the end of the string: `len(s) + i` or `len(s) + j` is substituted. But note that `-0` is still `0`.

4. The slice of *s* from *i* to *j* is defined as the sequence of items with index *k* such that `i <= k < j`. If *i* or *j* is greater than `len(s)`, use `len(s)`. If *i* is omitted or `None`, use `0`. If *j* is omitted or `None`, use `len(s)`. If *i* is greater than or equal to *j*, the slice is empty.

5. The slice of *s* from *i* to *j* with step *k* is defined as the sequence of items with index `x = i + n*k` such that `0 <= n < (j-i)/k`. In other words, the indices are `i`, `i+k`, `i+2*k`, `i+3*k` and so on, stopping when *j* is reached (but never including *j*). If *i* or *j* is greater than `len(s)`, use `len(s)`. If *i* or *j* are omitted or `None`, they become "end" values (which end depends on the sign of *k*). Note, *k* cannot be zero. If *k* is `None`, it is treated like `1`.

game

PYOGGLE

Example: s = “Singaporean”

s[0:3] = “Sin” ← *Not* “Sing” *!*

s[:5:-2] + s[-1] = “neon”

s[0] + s[1:3] * 2 = “Singing”

s[7] + s[1: :-1] + s[-3] = “riSe” ← *Not allowed !*

s[2: :4] + “-” + s = “non-Singaporean”

Rules

- Can use + and *
- Bonus points for using step
- Bonus points for words longer that 7 letters

Dictionary

supercalifragilisticexpialidocious

supercalifragilisticexpialidocious

/ˌsuːpəkalɪfradʒɪlɪstɪkˌɛkspɪalɪˈdəʊʃəs/

adjective *informal*

extraordinarily good; wonderful.
"the only word to characterize Kepler's discoveries was 'Supercalifragilisticexpialidocious'"

Click to listen

```
s='supercalifragilisticexpialidocious'
    0123456789111111111122222222223333
                        01234567890123456789012
```

PYOGGLE

time

Wordpads for Class/Group Discussion

February 17, 2018 17:22 by Tan Bang Choon David

Whole Class | Group 1 | Group 2 | Group 3 | Group 4