Python

Programming with Python

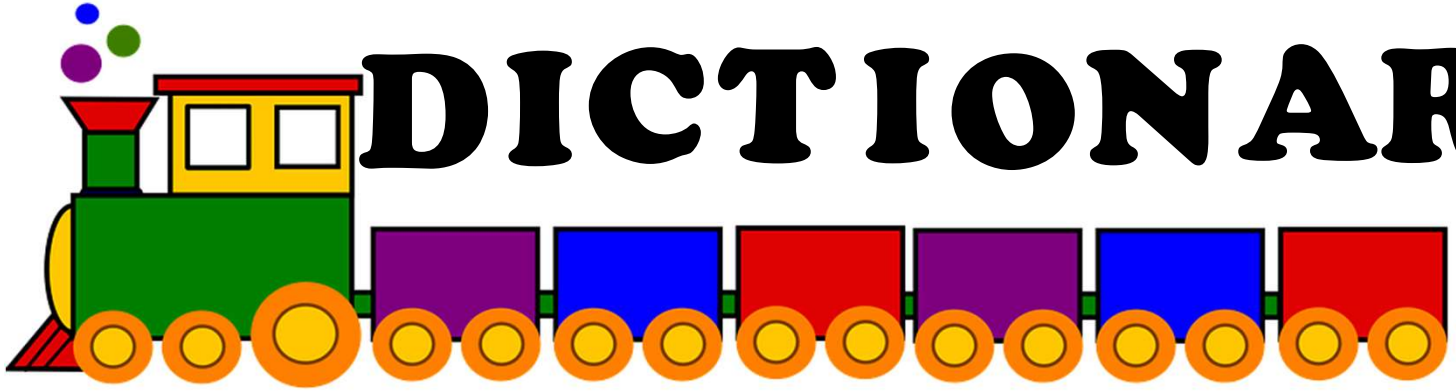
STRINGS

TUPLES

LISTS

SET

DICTIONARY



Strings

- Strings in Python have type str
- They represent sequence of characters
  - Python does not have a type corresponding to character.
- Strings are enclosed in single quotes(') or double quotes(“)
  - Both are equivalent
- Backslash (\) is used to escape quotes and special characters

Strings

```
>>> name='intro to python'
>>> descr='acad\'s first course'
```

- More readable when print is used

```
>>> print(descr)
acad's first course
```

Multiline string:

```
A= “ “ “ Geeksforgeeks, Noida
          An Ed-Tech company,
         founded by Mr Sandeep Jain ”  ” ”

>>> print(A)


Geeksforgeeks, Noida
An Ed-Tech company,
founded by Mr Sandeep Jain
```

Length of a String

- len function gives the length of a string

```
>>> name='intro to python'
>>> empty=''
>>> single='a'
```

\n is a **single** character: the special character representing newline

Concatenate and Repeat

- In Python, + and * operations have special meaning when operating on strings
  - + is used for concatenation of (two) strings
  - * is used to repeat a string, an int number of time
  - **Operator Overloading**

Concatenate and Repeat

```
>>> details = name + ', ' + descr
>>> details
"intro to python, acad's first course"
```

Indexing

- Strings can be indexed
- First character has index 0

```
>>> name='Acads'
```

Indexing

- Negative indices start counting from the right
- Negatives indices start from -1
- -1 means last, -2 second last, ...

```
>>> name='Acads'
>>> name[-1]
's'
>>> name[-5]
'A'
>>> name[-2]
'd'
```

Indexing

- Using an index that is too large or too small results in "index out of range" error

Looping Through a String

Since strings are arrays, we can loop through the characters in a string, with a `for` loop.

Example

Loop through the letters in the word “Hello World!”:

```
for x in "Hello World!":
>>> print(x)

H
e
l
l
o

.....
```

Membership role : in/ not in

Check String

To check if a certain phrase or character is present in a string, we can use the keyword in.

Example

Check if “best" is present in the following text:

```
>>>txt = “Honesty is the best policy!"
>>>print(“best" in txt)
```

True

Slicing

- To obtain a substring
- s[start:end] means substring of s starting at index start and ending at index end-1
- s[0:len(s)] is same as s
- Both start and end are optional
  - If start is omitted, it defaults to 0
  - If end is omitted, it defaults to the length of string
- s[:] is same as s[0:len(s)], that is same as s

Slicing

```
>>> name='Acads'
>>> name[0:3]
```

More Slicing

```
>>> name='Acads'
>>> name[-4:-1]
'cad'
>>> name[-4:]
'cads'
>>> name[-4:4]
'cad'
```

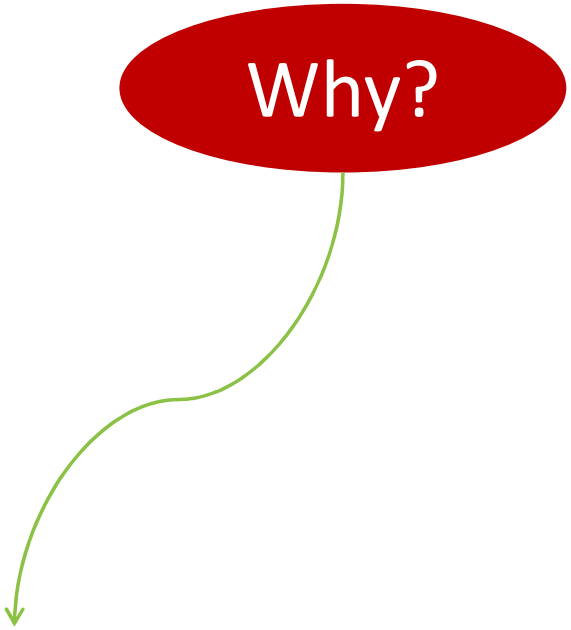
Understanding Indices for slicing

A	c	a	d	s	
0	1	2	3	4	5
-5	-4	-3	-2	-1	

Out of Range Slicing

A	c	a	d	s
0	1	2	3	4
-5	-4	-3	-2	-1

- Out of range indices are ignored for slicing
- when start and end have the same sign, if start >=end, empty slice is returned



Modifying String

```
a =  “Hello, World!”
b = “Hello World!          ”

a.upper()      #Upper case conversion
a.lower()      #Lower case Conversion
a.strip()      #Removes all whitespaces from string
               from both the ends of the string
a.replace("H","J", [count])
a.split(",")   #splits the string based on separator
```

f-strings

- A formatted string literal or f-string is a string literal that is prefixed with 'f' or 'F'.
- These strings may contain replacement fields, which are expressions delimited by curly braces { }.
- While other string literals always have a constant value, formatted strings are really expressions evaluated at run time.

```
>>> name = "Fred“
>>> f"He said his name is {name!r}.“
    "He said his name is 'Fred'.“

>>> f"He said his name is {repr(name)}." # repr() is equivalent to !r“
    He said his name is 'Fred'.“
```

Custom String Formatting

The built-in string class provides the ability to do complex variable substitutions and value formatting via the **format()** method

Three conversion flags are currently supported: '!s' which calls str() on the value, '!r' which calls repr() and '!a' which calls ascii().

Some examples:

```
"Harold's a clever {0!s}"        # Calls str() on the argument first
"Bring out the holy {name!r}"    # Calls repr() on the argument first
"More {!a}"                      # Calls ascii() on the argument first
```

Accessing arguments by position:

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{}, {}, {}'.format('a', 'b', 'c')  # 3.1+ only
'a, b, c'
```

Accessing arguments by name:

```
>>> 'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-115.81W')
'Coordinates: 37.24N, -115.81W'
```

Replacing `%s` and `%r`:

```
>>> "repr() shows quotes: {!r}; str() doesn't: {!s}".format('test1', 'test2')
"repr() shows quotes: 'test1'; str() doesn't: test2"
```

Accessing arguments' items:

```
>>> coord = (3, 5)
>>> 'X: {0[0]};  Y: {0[1]}'.format(coord)
'X: 3;  Y: 5'
```

Aligning the text and specifying a width:

```
>>> '{:<30}'.format('left aligned')
'left aligned                  '
>>> '{:>30}'.format('right aligned')
'                 right aligned'
>>> '{:^30}'.format('centered')
'           centered           '
>>> '{:*^30}'.format('centered')  # use '*' as a fill char
'***********centered***********'
```

Replacing `%+f`, `%-f`, and `% f` and specifying a sign:

```
>>> '{:+f}; {:+f}'.format(3.14, -3.14)  # show it always
'+3.140000; -3.140000'
>>> '{: f}; {: f}'.format(3.14, -3.14)  # show a space for positive numbers
' 3.140000; -3.140000'
>>> '{:-f}; {:-f}'.format(3.14, -3.14)  # show only the minus -- same as '{:f}; {:f}'
'3.140000; -3.140000'
```

More Functions on Python String

1. What is the output of the code shown below?

'{a}{b}{a}'.format(a='hello', b='world')

   A. 'hello world’
   B. 'hello’ 'world' ‘hello‘
   C. 'helloworldhello‘
   D. 'hello’ ‘hello’ ‘world'

2. What does this command do str[::- 1]

A. Selects only the last character of str
B. Selects full string except the last character
C. Reverses the str
D. Select the -1th character

Can you figure out the output?

```
1. s="Hello"
2. print(s.replace("l", "e"))
3. print(s[3])
4. print(len(s))
5. print(s.__getitem__(3))
```

Heeeo 1 2 5

Hello 1 2 5

Heeeo
l
5
l

Hello
1
2
3

After the assignment `signal = 'abracadabra'`, what is returned by `signal[len(signal)]`?

- 'a'
- 'abracadabra'
- 11
- an error

Determine the output for the following code

```
1. s = "Hi hElLo"
2. print s.upper()
3. print s.lower()
```

HI HELLO
hi hello

UPPER
LOWER

hi hello
HI HELLO

None of these