SNIST
Sreenidhi Institute of Science and Technology
Because, Life is all about taking the right decisions
AUTONOMOUS

PYTHON PROGRAMMING

Lecture 2 Unit 2

STRINGS

Name: Mr. Dheeraj Sundaragiri
Assistant Professor
Department of Computer Science and Engineering
SNIST

STRINGS

- Accessing Strings
- Basic Operations
- String slices
- Function and Methods

STRINGS INTRODUCTION

- A string is a sequence of characters.
- Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes.
- Creating strings is as simple as assigning a value to a variable.

STRINGS INTRODUCTION

- When a string contains numbers it is still a string.

  For example:

```
var1 = 'Hello World!'
var2 = "Python Programming"
var3 = """Python
    Programming"""
```

ACCESSING STRINGS

- Python does not support a character type; these are treated as strings of length one thus also considered a substring.
- To access substrings use the square brackets for slicing along with the index or indices to obtain your substring.

ACCESSING STRINGS

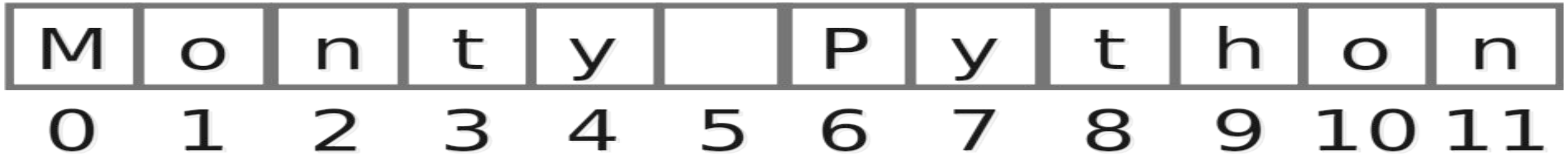
- Example:
  - var1 = 'Hello World!'
  - var2 = "Python Programming"
  - print "var1[0]: " var1[0]
  - print "var2[1:5]: " var2[1:5]
- This will produce following result:
  - var1[0]: H
  - var2[1:5]: ytho

STRING OPERATIONS

- concatenate with + or neighbors
  - word = "Help" + "x"
  - word = "Help" "a"
- subscripting of strings
  - Hello[2] l
  - slice: Hello[1:2] el
  - word[-1] last character
  - len(word) 5
- immutable: cannot assign to subscript

SLICING STRINGS

```
>>> str = 'Monty Python'
```



- The string position starts at zero
- We can also look at any continuous section of a string using a colon operator

```
>>> print str[0:4] # Mont
```

SLICING STRINGS

- The second number is one beyond the end of the slice - “up to but not including”

```
>>> print str[6:7] # P
```

- If the second number is beyond the end of the string it stops at the end

```
>>> print str[6:20] # Python
```

- Slicing reverse

```
>>> print(str[-6:]) # Python
```

STRINGS: Functions and Methods

- Python has a number of string functions which are in the string library
- These functions are already built into every string - we invoke them by appending the function to the string variable

```
var1 = 'Hello World!'
dir(var1)
```

STRINGS: Functions and Methods

- These functions do not modify the original string instead they return a new string that has been altered

1. capitalize
2. casefold
3. center
4. count
5. encode
6. endswith
7. expandtabs
8. find
9. format
10. format_map
11. index
12. isalnum
13. isalpha
14. isdecimal
15. isdigit

STRINGS: Functions and Methods

16. isidentifier
17. islower
18. isnumeric
19. isprintable
20. isspace
21. istitle
22. isupper
23. join
24. ljust
25. lower
26. lstrip
27. maketrans
28. partition
29. replace
30. rfind
31. rindex
32. rjust
33. rpartition
34. rsplit
35. rstrip
36. split
37. splitlines
38. startswith
39. strip
40. swapcase
41. title
42. translate
43. upper
44. zfill

STRINGS: Methods 1 of 20

1. capitalize
2. casefold
3. center
4. count
5. encode
6. endswith
7. expandtabs
8. find
9. format
10. format_map
11. index
12. isalnum
13. isalpha
14. isdecimal
15. isdigit

```
>>> str="hello"
>>> str.capitalize()  #  'Hello'
>>> str="hello PYTHON"
>>> str.casefold()# 'hello python'
>>> str.center(24)    # width
'      hello PYTHON      '
>>> str.center(24,'*') # fillchar
'******hello PYTHON******'
```

STRINGS: Methods 2 of 20

1. ~~capitalize~~
2. ~~casefold~~
3. ~~center~~
4. count
5. encode
6. endswith
7. expandtabs
8. find
9. format
10. format_map
11. index
12. isalnum
13. isalpha
14. isdecimal
15. isdigit

```
>>> str="hello PYTHON"
>>> str.count('o') #substring : 1
>>> str.count('O',0,5)
# start, end : 0
>>> string = 'pythön!'
# string.endswith(suffix[, start[, end]])
>>> string.endswith('!') #    True
>>> string.endswith('.') #    False
```

STRINGS: Methods 3 of 20

1. ~~capitalize~~
2. ~~casefold~~
3. ~~center~~
4. ~~count~~
5. encode
6. ~~endswith~~
7. expandtabs
8. find
9. format
10. format_map
11. index
12. isalnum
13. isalpha
14. isdecimal
15. isdigit

```
>>> string = 'pythön!'
# encode to default Utf-8 Encoding
>>> string.encode() b'pyth\xc3\xb6n!'
# ignore error
>>> string.encode("ascii", "ignore")
    b'pythn!'
# replace error
>>> string.encode("ascii", "replace")
    b'pyth?n!'
```

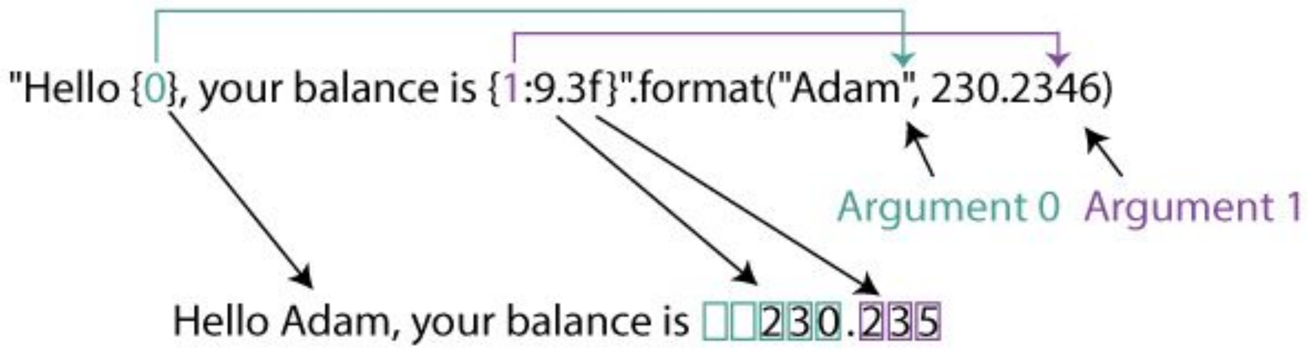
STRINGS: Methods 4 of 20

1. ~~capitalize~~
2. ~~casefold~~
3. ~~center~~
4. ~~count~~
5. ~~encode~~
6. ~~endswith~~
7. expandtabs
8. find
9. format
10. format_map
11. index
12. isalnum
13. isalpha
14. isdecimal
15. isdigit

```
>>> str="Hi i am in\t#INDIA"
>>> str.expandtabs() # tab size 8
'Hi i am in      #INDIA'
>>> str.expandtabs(2) # tab size 2
'Hi i am in  #INDIA'
>>> data=str.find('#')
>>> data             #  11
>>> str[data+1:]     #  INDIA
```

STRINGS: Methods 5 of 20

1. ~~capitalize~~
2. ~~casefold~~
3. ~~center~~
4. ~~count~~
5. ~~encode~~
6. ~~endswith~~
7. ~~expandtabs~~
8. ~~find~~
9. format
10. format_map
11. index
12. isalnum
13. isalpha
14. isdecimal
15. isdigit


"Hello {0}, your balance is {1:9.3f}".format("Adam", 230.2346)
Argument 0
Argument 1
Hello Adam, your balance is 230.235

```
>>> str="Hi {0}, Today is
{1}".format("PYTHON","Good Day")
>>> str
'Hi PYTHON, Today is Good Day'
```

STRINGS: Methods 6 of 20

1. ~~capitalize~~
2. ~~casefold~~
3. ~~center~~
4. ~~count~~
5. ~~encode~~
6. ~~endswith~~
7. ~~expandtabs~~
8. ~~find~~
9. ~~format~~
10. format_map
11. index
12. isalnum
13. isalpha
14. isdecimal
15. isdigit

```
>>> point = {'x':4,'y':-5}
>>> print('{x} {y}'.format(**point))
>>> print('{x} {y}'.format_map(point))
 # Both gives same output 4 -5
 # str.format(**mapping) copies the dict
 # str.format_map(mapping) makes a new dictionary during method call.
```

STRINGS: Methods 7 of 20

1. ~~capitalize~~
2. ~~casefold~~
3. ~~center~~
4. ~~count~~
5. ~~encode~~
6. ~~endswith~~
7. ~~expandtabs~~
8. ~~find~~
9. ~~format~~
10. ~~format_map~~
11. index
12. isalnum
13. isalpha
14. isdecimal
15. isdigit

# str.index(sub[, start[, end]])

```
>>> str = 'Hi i am in\t#INDIA'
>>> str.index('in')          #  8
>>> str.index('in',8)        #  8
>>> str.index('IN',8,-3)  #  12
>>> str.index('i',2,30)          #  3
>>> str[1:2].isalnum()    #  True
>>> str[1:2].isalpha()#  True
>>> str.isalpha()            #  False
```

STRINGS: Methods 8 of 20

~~11. index~~
~~12. isalnum~~
~~13. isalpha~~
14. isdecimal
15. isdigit
16. isidentifier
17. islower
18. isnumeric
19. isprintable
20. isspace
21. istitle
22. isupper
23. join
24. ljust
25. lower

```
>>> str = 'Hi i am in\t#INDIA'
>>> str.isdecimal()          #   False
>>> str.isdigit()            #   False
# The superscript and subscripts are
considered digit characters but not
decimals.
>>> str[9:10].isidentifier() #   True
>>> str[1:-5].islower()      #   True
```

STRINGS: Methods 9 of 20

```
str = '1242323'
print(str.isnumeric())        #  True
#str = '²3455'
str = '\u00B23455'
print(str.isnumeric())        #  True
# str = '½'
str = '\u00BD'
print(str.isnumeric())        #  True
s='python12'
print(s.isnumeric())        #  False
```

STRINGS: Methods 10 of 20

19. isprintable
20. isspace
21. istitle
22. isupper
23. join
24. ljust
25. lower
26. lstrip
27. maketrans
28. partition
29. replace
30. rfind
31. rindex
32. rjust
33. rpartition

```
>>> str = ' '
>>> str.isprintable() #   True
>>> str = '\n'
>>> str.isprintable() #   False
>>> str = ''
>>> str.isprintable())#   True
>>> str = '   \t'
>>> str.isspace()        #   True
>>> str = 'Monty @ Python'
>>> str.istitle()      #   True
```

STRINGS: Methods 11 of 20

19. ~~isprintable~~
20. ~~isspace~~
21. ~~istitle~~
22. isupper
23. join
24. ljust
25. lower
26. lstrip
27. maketrans
28. partition
29. replace
30. rfind
31. rindex
32. rjust
33. rpartition

```
>>> str = 'Monty @ Python'
>>> str.isupper()        #  False
>>> lst=['1','2','3','4','5']
>>> ch='-'
>>> ch.join(lst)      #  '1-2-3-4-5'
>>> str.ljust(20,'*')
'Monty @ Python******'
>>> str.lower()
'monty @ python'
```

STRINGS: Methods 12 of 20

26. lstrip
27. maketrans
28. partition
29. replace
30. rfind
31. rindex
32. rjust
33. rpartition
34. rsplit
35. rstrip
36. split
37. splitlines
38. startswith
39. strip
40. swapcase

```
>>> str = '   Monty @ Python   '
>>> str.lstrip()
'Monty @ Python   '
>>> str.rstrip()
'   Monty @ Python'
>>> str.strip()
'Monty @ Python'
>>> str.strip().strip('on')
'Monty @ Pyth'
```

STRINGS: Methods 13 of 20

27. maketrans
28. partition
29. replace
30. rfind
31. rindex
32. rjust
33. rpartition
34. rsplit
~~35. rstrip~~
36. split
37. splitlines
38. startswith
~~39. strip~~
40. swapcase
41. title
42. translate
43. upper
44. zfill

```
>>> str = 'Monty @ Python'
>>> str.rjust(20,'*')
'******Monty @ Python'
>>> str.replace("@","Hello")
'Monty Hello Python'
>>> str.swapcase()
'mONTY @ pYTHON'
>>> str.upper()
'MONTY @ PYTHON'
```

STRINGS: Methods 14 of 20

27. maketrans
28. partition
~~29. replace~~
30. rfind
31. rindex
~~32. rjust~~
33. rpartition
34. rsplit
~~35. rstrip~~
36. split
37. splitlines
38. startswith
~~39. strip~~
~~40. swapcase~~
41. title
42. translate
~~43. upper~~
44. zfill

# maketrans() creates a mapping of the character's Unicode ordinal to its corresponding translation.

```
>>> str1 = "aeiou"
>>> str2 = "12345" #must have
same len
>>> str.maketrans(str1, str2)
{97: 49, 101: 50, 105: 51, 111: 52,
117: 53}
```

STRINGS: Methods 15 of 20

27. maketrans
28. partition
~~29. replace~~
30. rfind
31. rindex
~~32. rjust~~
33. rpartition
34. rsplit
~~35. rstrip~~
36. split
37. splitlines
38. startswith
~~39. strip~~
~~40. swapcase~~
41. title
42. translate
~~43. upper~~
44. zfill

```
>>> strt = str.maketrans(str1, str2)
>>> str = "this is string
example....wow!!!"
>>> print (str.translate(strt))
 th3s 3s str3ng 2x1mpl2....w4w!!!
>>> str3 = "ae"
>>> strt = str.maketrans(str1, str2,
str3)
>>> print (str.translate(strt))
 th3s 3s str3ng xmpl....w4w!!!
```

STRINGS: Methods 16 of 20

28. partition
29. ~~replace~~
30. rfind
31. rindex
32. ~~rjust~~
33. rpartition
34. rsplit
35. ~~rstrip~~
36. split
37. splitlines
38. startswith
39. ~~strip~~
40. ~~swapcase~~
41. title
42. ~~translate~~

```
>>> str = 'Monty @ Python'
>>> str.partition("@") # creates tuple
('Monty ', '@', ' Python')
>>> str.rpartition("on")
# at last occurrence
('Monty @ Pyth', 'on', '')
>>> str.rfind('on')
# returns highest index 12
```

STRINGS: Methods 17 of 20

~~28. partition~~
~~29. replace~~
~~30. rfind~~
31. rindex
~~32. rjust~~
~~33. rpartition~~
34. rsplit
~~35. rstrip~~
36. split
37. splitlines
38. startswith
~~39. strip~~
~~40. swapcase~~
41. title
~~42. translate~~

```
>>> str = 'Monty @ Python'
>>> str.rindex('n')
# returns highest index 13
# rfind() returns -1 if the substring is not found, whereas rindex() throws an exception.
>>> str='hello monty'
>>> str.title()     # 'Hello Monty'
```

STRINGS: Methods 18 of 20

28. ~~partition~~
29. ~~replace~~
30. ~~rfind~~
31. ~~rindex~~
32. ~~rjust~~
33. ~~rpartition~~
34. rsplit
35. ~~rstrip~~
36. split
37. splitlines
38. startswith
39. ~~strip~~
40. ~~swapcase~~
41. ~~title~~
42. ~~translate~~

```
>>> str = 'Monty @ Python'
>>> str.split("@")
# ['Monty ', ' Python']
>>> str.rsplit("@")
# Same output, but Fractional Slow
>>> str.split(" ",1)
# ['Monty', '@ Python']
>>> str.rsplit(" ",1)
# ['Monty @', 'Python']
```

STRINGS: Methods 19 of 20

31. ~~rindex~~
32. ~~rjust~~
33. ~~rpartition~~
34. ~~rsplit~~
35. ~~rstrip~~
36. ~~split~~
37. splitlines
38. startswith
39. ~~strip~~
40. ~~swapcase~~
41. ~~title~~
42. ~~translate~~
43. ~~upper~~
44. zfill

```
>>> str='Monty\n@\rPython'
>>> str.splitlines()
['Monty', '@', 'Python']
>>> str.splitlines(True)
['Monty\n', '@\r', 'Python']
>>> str='Monty @ Python'
>>> str.splitlines()
['Monty @ Python']
>>> str.startswith('Mon')    # True
```

STRINGS: Methods 20 of 20

31. ~~rindex~~
32. ~~rjust~~
33. ~~rpartition~~
34. ~~rsplit~~
35. ~~rstrip~~
36. ~~split~~
37. ~~splitlines~~
38. startswith
39. ~~strip~~
40. ~~swapcase~~
41. ~~title~~
42. ~~translate~~
43. ~~upper~~
44. zfill

```
>>> str='Monty\n@\rPython'
>>> str.startswith('Mon')     # True
>>> str.startswith('Tue') # False
>>> num="123"
>>> num.zfill(8)
'00000123'
>>> num.zfill(8)      # num= "+123"
'+0000123'
```

The topic of next lecture is

- Lists
- Tuples
- Dictionaries