Numbers and Strings

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy/Python

Number Data Types

- Store numeric values.
- They are ***immutable*** data types.
- This means, changing the value of a number data type results in a newly allocated object.
- Number objects are created when you assign a value to them.

```
var1 = 1          var2 = 10
```

- Can also delete the reference to a number object by using the `del` statement.

```
del var1, var2
```

Number Data Types (Cont.)

- `int` (signed integers)
  - Are often called just integers or ints.
  - Are positive or negative whole numbers with no decimal point.
  - Integers in Python 3 are of unlimited size.
  - Python 2 has two integer types - `int` and `long`.
  - There is no ***long integer*** in Python 3 anymore.

Number Data Types (Cont.)

- `float` (floating point real values)
  - Called floats.
  - Represents real numbers.
  - Are written with a decimal point dividing the integer and the fractional parts.
  - Floats may be in scientific notation, with `E` or `e` indicating the power of 10 (2.5e2 = 2.5 x 10^2 = 250).

Number Data Types (Cont.)

- **`complex`** (complex numbers)
  - Are of the form **`a + bJ`**.
  - Where **`a`** and **`b`** are floats and **`J`** (or **`j`**) represents the square root of -1 (which is an imaginary number).
  - The ***real*** part of the number is **`a`**, and the ***imaginary*** part is **`b`**.
  - Complex numbers are not used much in Python programming.

Number Type Conversion

- Type `int(x)` to convert `x` to a plain integer.
- Type `long(x)` to convert `x` to a long integer.
- Type `float(x)` to convert `x` to a floating-point number.
- Type `complex(x)` to convert `x` to a complex number with real part `x` and imaginary part zero.
- Type `complex(x, y)` to convert `x` and `y` to a complex number with real part `x` and imaginary part `y`.
- `x` and `y` are numeric expressions.

Mathematical Functions

- **`abs(x)`**
  - Returns the absolute value of **`x`**.
  - i.e. the positive distance between **`x`** and zero.

```
>>> print ("abs(-45) : ", abs(-45))
abs(-45) :  45
>>>
>>> print ("abs(100.12) : ", abs(100.12))
abs(100.12) :  100.12
>>>
```

Mathematical Functions (Cont.)

- **`ceil(x)`**
  - Returns the ceiling value of **`x`**.
  - i.e. the smallest integer not less than **`x`**.

```
>>> import math    # This wil import math module
>>>
>>> print ("math.ceil(-45.17) : ", math.ceil(-45.17))
math.ceil(-45.17) :  -45
>>> print ("math.ceil(100.12) : ", math.ceil(100.12))
math.ceil(100.12) :  101
>>> print ("math.ceil(100.72) : ", math.ceil(100.72))
math.ceil(100.72) :  101
>>> print ("math.ceil(math.pi) : ", math.ceil(math.pi))
math.ceil(math.pi) :  4
>>>
```

Mathematical Functions (Cont.)

- `cmp(x, y)`
  - `-1` if x < y
  - `0` if x == y
  - `1` if x > y.
  - *Deprecated* in Python 3.
  - Instead use return `(x > y) - (x < y)`.

Mathematical Functions (Cont.)

- **exp(x)**
  - Returns exponential of **x**.
  - e^x.

```
>>> import math
>>>
>>> print ("math.exp(-45.17) : ", math.exp(-45.17))
math.exp(-45.17) :  2.4150062132629406e-20
>>> print ("math.exp(100.12) : ", math.exp(100.12))
math.exp(100.12) :  3.0308436140742566e+43
>>> print ("math.exp(100.72) : ", math.exp(100.72))
math.exp(100.72) :  5.522557130248187e+43
>>> print ("math.exp(math.pi) : ", math.exp(math.pi))
math.exp(math.pi) :  23.14069263277926
>>>
>>> print ("math.exp(5) : ", math.exp(5))
math.exp(5) :  148.4131591025766
>>>
```

Mathematical Functions (Cont.)

- **`fabs(x)`**
  - Returns the absolute value of **`x`**.
  - Although similar to the **`abs()`** function, there are differences between the two functions. They are :
    - **`abs()`** is a built in function whereas **`fabs()`** is defined in **`math`** module.
    - **`fabs()`** function works only on float and integer whereas **`abs()`** works with complex number also.

```
>>> import math
>>>
>>> print ("math.fabs(-45.17) : ", math.fabs(-45.17))
math.fabs(-45.17) :  45.17
>>> print ("math.fabs(math.pi) : ", math.fabs(math.pi))
math.fabs(math.pi) :  3.141592653589793
>>>
```

Mathematical Functions (Cont.)

- **`floor(x)`**
  - Returns the floor of **`x`**.
  - i.e. the largest integer not greater than **`x`**.

```
>>> import math
>>>
>>> print ("math.floor(-45.17) : ", math.floor(-45.17))
math.floor(-45.17) :  -46
>>> print ("math.floor(100.12) : ", math.floor(100.12))
math.floor(100.12) :  100
>>> print ("math.floor(100.72) : ", math.floor(100.72))
math.floor(100.72) :  100
>>> print ("math.floor(math.pi) : ", math.floor(math.pi))
math.floor(math.pi) :  3
>>>
```

Mathematical Functions (Cont.)

- **`log(x)`**
  - Returns the natural logarithm of **`x`**, for x > 0.

```
>>> import math
>>>
>>> print ("math.log(100.12) : ", math.log(100.12))
math.log(100.12) :  4.606369465635735
>>> print ("math.log(100.72) : ", math.log(100.72))
math.log(100.72) :  4.612344389736092
>>> print ("math.log(math.pi) : ", math.log(math.pi))
math.log(math.pi) :  1.1447298858494002
>>>
```

Mathematical Functions (Cont.)

- **`log10(x)`**
  - Returns base-10 logarithm of **`x`**, for x > 0.

```
>>> import math
>>>
>>> print ("math.log10(100.12) : ", math.log10(100.12))
math.log10(100.12) :  2.000520840936185 4
>>> print ("math.log10(100.72) : ", math.log10(100.72))
math.log10(100.72) :  2.003115717099806
>>>
>>> print ("math.log10(119) : ", math.log10(119))
math.log10(119) :  2.07554696139253 1
>>> print ("math.log10(math.pi) : ", math.log10(math.pi))
math.log10(math.pi) :  0.497149872694133 8
>>>
```

Mathematical Functions (Cont.)

- **`max(x1, x2, …)`**
  - Returns the largest of its arguments.
  - i.e. the value closest to positive infinity.

```
>>> print ("max(80, 100) : ", max(80, 100))
max(80, 100) :  100
>>> print ("max(-20, 100 ,400) : ", max(-20, 100, 400))
max(-20, 100 ,400) :  400
>>> print ("max(-80, -20, -10) : ", max(-80, -20, -10))
max(-80, -20, -10) :  -10
>>> print ("max(0, 100, -400) : ", max(0, 100, -400))
max(0, 100, -400) :  100
>>>
```

Mathematical Functions (Cont.)

- **`min(x1, x2, …)`**
  - Returns the smallest of its arguments.
  - i.e. the value closest to negative infinity.

```
>>> print ("min(80, 100, 1000) : ", min(80, 100, 1000))
min(80, 100, 1000) :  80
>>> print ("min(-20, 100, 400) : ", min(-20, 100, 400))
min(-20, 100, 400) :  -20
>>> print ("min(-80, -20, -10) : ", min(-80, -20, -10))
min(-80, -20, -10) :  -80
>>>
>>> print ("min(0 ,100, -400) : ", min(0, 100, -400))
min(0 ,100, -400) :  -400
>>>
```

Mathematical Functions (Cont.)

- **`modf(x)`**
  - Returns the fractional and integer parts of **`x`** in a two-item tuple.
  - Both parts have the same sign as **`x`**.
  - The integer part is returned as a **`float`**.

```
>>> import math
>>>
>>> print ("math.modf(100.12) : ", math.modf(100.12))
math.modf(100.12) :  (0.12000000000000455, 100.0)
>>> print ("math.modf(100.72) : ", math.modf(100.72))
math.modf(100.72) :  (0.7199999999999989, 100.0)
>>> print ("math.modf(119) : ", math.modf(119))
math.modf(119) :  (0.0, 119.0)
>>> print ("math.modf(math.pi) : ", math.modf(math.pi))
math.modf(math.pi) :  (0.14159265358979312, 3.0)
>>>
```

Mathematical Functions (Cont.)

- **`pow(x, y)`**
  - Returns the value of x^y.

```
>>> import math
>>>
>>>
>>> print ("math.pow(100, 2)) : ", math.pow(100, 2))
math.pow(100, 2)) :  10000.0
>>> print ("math.pow(100, -2)) : ", math.pow(100, -2))
math.pow(100, -2)) :  0.0001
>>> print ("math.pow(2, 4) : ", math.pow(2, 4))
math.pow(2, 4) :  16.0
>>> print ("math.pow(3, 0) : ", math.pow(3, 0))
math.pow(3, 0) :  1.0
>>>
```

Mathematical Functions (Cont.)

- **round(x, [,n])**
  - Is a built-in function in Python.
  - It returns **x** rounded to **n** digits from the decimal point.

```
>>> print ("round(70.23456) : ", round(70.23456))
round(70.23456) :  70
>>> print ("round(56.659, 1) : ", round(56.659, 1))
round(56.659, 1) :  56.7
>>> print ("round(80.264, 2) : ", round(80.264, 2))
round(80.264, 2) :  80.26
>>> print ("round(100.000056, 3) : ", round(100.000056, 3))
round(100.000056, 3) :  100.0
>>> print ("round(-100.000056, 3) ", round(-100.000056, 3))
round(-100.000056, 3)  -100.0
>>>
```

Mathematical Functions (Cont.)

- **`sqrt(x)`**
  - Returns the square root of **`x`** for x > 0.

```
>>> import math
>>>
>>> print ("math.sqrt(100) : ", math.sqrt(100))
math.sqrt(100) :  10.0
>>> print ("math.sqrt(7) : ", math.sqrt(7))
math.sqrt(7) :  2.6457513110645907
>>> print ("math.sqrt(math.pi) : ", math.sqrt(math.pi))
math.sqrt(math.pi) :  1.7724538509055159
>>>
```

Random Number Functions

- Random numbers are used for games, simulations, testing, security, and privacy applications.
- **`choice(seq)`**
  - Returns a random item from a list, tuple, or string.

```
>>> import random
>>>
>>> print ("returns a random number from range(100) : ", random.choice(range(100)))
returns a random number from range(100) :  71
>>> print ("returns random element from list [1,2,3,5,9]) : ", random.choice([1,2,3,5,9]))
returns random element from list [1,2,3,5,9]) :  9
>>> print ("returns random character from string 'Hello World' : ", random.choice('Hello, World'))
returns random character from string 'Hello World' :  e
>>>
```

Random Number Functions (Cont.)

- **`randrange([start,] stop [, step])`**
  - Returns a randomly selected element from **`range(start, stop, step)`**.

```
>>> import random
>>>
>>> # randomly select an odd number between 1-100
>>> print ("randrange(1, 100, 2) : ", random.randrange(1, 100, 2))
randrange(1, 100, 2) :  1
>>>
>>> # randomly select a number between 0-99
>>> print ("randrange(100) : ", random.randrange(100))
randrange(100) :  8
>>>
```

Random Number Functions (Cont.)

- **`random()`**
  - Returns a random floating point number in the range [0.0, 1.0].

```
>>> import random
>>>
>>> print ("random() : ", random.random())
random() :  0.7342871419551999
>>>
>>> print ("random() : ", random.random())
random() :  0.2785271852536648
>>>
```

Random Number Functions (Cont.)

- **`seed([x])`**
  - Initializes the basic random number generator.
  - Call this function before calling any other random module function.

```
>>> import random
>>>
>>> random.seed()
>>> print ("random number with default seed", random.random())
random number with default seed 0.642248299140799
>>> random.seed(10)
>>> print ("random number with int seed", random.random())
random number with int seed 0.5714025946899135
>>> random.seed("hello", 2)
>>> print ("random number with string seed", random.random())
random number with string seed 0.3537754404730722
>>>
```

Random Number Functions (Cont.)

- **`shuffle(lst)`**
  - Randomizes the items of a list in place.

```
>>> import random
>>>
>>> list = [20, 16, 10, 5]
>>> random.shuffle(list)
>>> print ("Reshuffled list : ", list)
Reshuffled list :  [16, 10, 20, 5]
>>>
>>> random.shuffle(list)
>>> print ("Reshuffled list : ", list)
Reshuffled list :  [20, 16, 10, 5]
>>>
```

Random Number Functions (Cont.)

- **`uniform(x, y)`**
  - Returns a random float **`r`**, such that **`x`** is less than or equal to **`r`** and **`r`** is less than **`y`**.

```
>>> import random
>>>
>>> print ("Random Float uniform(5, 10) : ", random.uniform(5, 10))
Random Float uniform(5, 10) :  9.390002601406866
>>>
>>> print ("Random Float uniform(7, 14) : ", random.uniform(7, 14))
Random Float uniform(7, 14) :  8.937793345962156
>>>
```

Mathematical Constants

- `pi`
  - The mathematical constant `pi`.
- `e`
  - The mathematical constant `e`.

Numbers - Lab

```
1 a = 11113
2 b = 23
3 ret1, ret2 = divmod(a, b)
4 print('<%d/%d>는 몫이 <%d>, 나머지가 <%d>입니다.' %(a, b, ret1, ret2))
5
```

Numbers – Lab (Cont.)

```
1 abs1 = abs(-3)          # 정수의 절대값
2 abs2 = abs(-5.72)       # 실수의 절대값
3 abs3 = abs(3+4j)        # 복소수의 절대값
4 print(abs1)         # 3이 출력됨
5 print(abs2)         # 5.72가 출력됨
6 print(abs3)         # 5.0이 출력됨
7
```

Numbers – Lab (Cont.)

```
1 ret1 = round(1118)     # 소수점 첫째자리에서 반올림해줌
2 ret2 = round(16.554)    # 소수점 첫째자리에서 반올림해줌
3 ret3 = round(1118, -1)  # 1자리에서 반올림해줌
4 ret4 = round(16.554, 2)  # 소수점 셋째자리에서 반올림해줌
5 print(ret1)   # 1118
6 print(ret2)   # 17
7 print(ret3)   # 1120
8 print(ret4)   # 16.55
9
```

Numbers – Lab (Cont.)

```
1 idata1 = int(-5.4)
2 idata2 = int(1.78e1)
3 idata3 = int(171.56)
4 print(idata1)    # -5가 출력됨
5 print(idata2)    # 17이 출력됨
6 print(idata3)    # 171이 출력됨
7
```

Strings Data Types

- Are amongst the most popular types in Python.
- 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.

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

Accessing Values in 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.

```
>>> var1= 'Hello World!'
>>> var2 = "Python Programming"
>>>
>>> print ("var1[0] = ", var1[0])
var1[0] =  H
>>> print ("var2[1:5] = ", var2[1:5])
var2[1:5] =  ytho
```

Updating Strings

- Can ***update*** an existing string by (re)assigning a variable to another string.
- The new value can be related to its previous value or to a completely different string altogether.

```
>>> var1 = 'Hello World!'
>>>
>>> print ("Updated String : - ", var1[:6] + ' Python')
Updated String : -  Hello  Python
>>>
```

Escape Characters

Baskslash notation	Hexadecimal character	Description
`\a`	0x07	Bell or alert
`\b`	0x08	Backspace
`\cx`		Control-x
`\C-x`		Control-x
`\e`	0x1b	Escape
`\f`	0x0c	Formfeed
`\M-\C-x`		Meta-Control-x
`\n`	0x0a	Newline

Escape Characters (Cont.)

Basckslash notation	Hexadecimal character	Description
`\nnn`		Octal notation, where `n` is in the range 0, 7
`\r`	0x0d	Carriage return
`\s`	0x20	Space
`\t`	0x09	Tab
`\v`	0x0b	Vertical tab
`\x`		Character x
`\xnn`		Hexadecimal notation, where `n` is in the range 0,9 a, f or A, F

String Special Operators

Operator	Description	Example
`+`	Concatenation - Adds values on either side of the operator	a `+` b will give `HelloPython`
`*`	Repetition - Creates new strings, concatenating multiple copies of the same string	a `*` 2 will give - `HelloHello`
`[]`	Slice - Gives the character from the given index	a`[1]` will give `e`
`[ : ]`	Range Slice - Gives the characters from the given range	a`[1:4]` will give `ell`

String Special Operators (Cont.)

Operator	Description	Example
`in`	Membership - Returns true if a character exists in the given string	`H` in a will give `1`
`not in`	Membership - Returns true if a character does not exist in the given string	`M` not in a will give `1`
`%`	Format - Performs String formatting	See at next slide

String Formatting Operator

- One of Python's coolest features is the string format operator %.
- This operator is unique to strings and makes up for the pack of having functions from C's *printf()* family.

```
>>> print ("My name is %s and weight is %d kg!" % ('Zara', 21))
My name is Zara and weight is 21 kg!
>>>
```

String Formatting Operator (Cont.)

Format Symbol & Conversion	Description
`%c`	Character
`%s`	String conversion via str() prior to formatting
`%i`	Signed decimal integer
`%d`	Signed decimal integer
`%u`	Unsinged decimal integer
`%o`	Octal integer
`%x`	Hexadecimal integer (lowercase letters)

String Formatting Operator (Cont.)

Format Symbol & Conversion	Description
`%X`	Hexadecimal integer(UPPERcase letters)
`%e`	Exponential notation (with lowercase ‘e’)
`%E`	Exponential notation (with UPPERcase ‘E’)
`%f`	Floating point real number
`%g`	The shorter of `%f` and `%e`
`%G`	The shorter of `%f` and `%E`

String Formatting Operator (Cont.)

- Other supported symbols and functionality are listed in the following table :

Symbol & Functionality	Description
*	Argument specifies width or precision
-	Left justification
+	Display the sign
<sp>	Leave a blank space before a positive number
#	Add the octal leading zero('0') or hexadecimal leading '0x' or '0X', depending on whether 'x' or 'X' were used.

String Formatting Operator (Cont.)

- Other supported symbols and functionality are listed in the following table :

Symbol & Functionality	Description
`0`	Pad from left with zeros (instead of spaces)
`%`	'%%' leaves you with a single literal '%'
`(var)`	Mapping variable (dictionary arguments)
`m.n.`	`m` is the minimum total width and `n` is the number of digits to display after the decimal point (if appl.)

Triple Quotes

- Python's triple quotes comes to the rescue by allowing strings to span multiple lines, including verbatim NEWLINEs, TABs, and any other special characters.

```
>>> para_str = """this is a long string that is made up of
several lines and mon-printable characters such as
TAB(₩t) and they will show up that way when displayed.
NEWLINEs within the string, wheter explicitly given like
this within the brackets [₩n], or just a NEWLINE within
the variable assignment will also show up.
"""
>>> print (para_str)
this is a long string that is made up of
several lines and mon-printable characters such as
TAB(        ) and they will show up that way when displayed.
NEWLINEs within the string, wheter explicitly given like
this within the brackets [
], or just a NEWLINE within
the variable assignment will also show up.

>>>
```

Triple Quotes (Cont.)

- Raw strings do not treat the backslash as a special character at all.
- Every character put into a raw string stays the way you wrote it :

```
>>>
>>> print ('C:₩₩nowhere')
C:₩nowhere
>>>
>>> print (r'C:₩₩nowhere')
C:₩₩nowhere
>>>
```

Unicode String

- In Python 3, all strings are represented in Unicode.
- In Python 2 are stored internally as 8-bit ASCII, hence it is required to attach 'u' to make it Unicode.
- It is no longer necessary now.

Built-in String Methods

- `capitalize()`
  - Capitalizes first letter of string

Built-in String Methods (Cont.)

- **`center(witdh,fillchar)`**
  - Returns a string padded with **`fillchar`** with the original string centered to a total of **`width`** columns.

Built-in String Methods (Cont.)

- `count(str, beg = 0, end = len(string))`
  - Counts how many times `str` occurs in string or in a substring of string if starting index `beg` and ending index `end` are given.

Built-in String Methods (Cont.)

- `decode(encoding = 'UTF-8', errors = 'strict')`
  - Decodes the string using the codec registered for encoding. `encoding` defaults to the default string encoding.

Built-in String Methods (Cont.)

- `encode(encoding = 'UTF-8', errors = 'strict')`
  - Returns encoded string version of string; on error, default is to raise a `ValueError` unless `errors` is given with 'ignore' or 'replace'.

Built-in String Methods (Cont.)

- `endswith(suffix, beg = 0, end = len(string))`
  - Determines if string or a substring of string (if starting index `beg` and ending index end are given) ends with `suffix`; returns `true` if so and `false` otherwise.

Built-in String Methods (Cont.)

- `expandtabs(tabsize = 8)`
  - Expands tabs in string to multiple spaces; defaults to `8` spaces per tab if `tabsize` not provided.

Built-in String Methods (Cont.)

- `find(str, beg = 0, end = len(string))`
  - Determine if `str` occurs in string or in a substring of string if starting index `beg` and ending index `end` are given returns index if found and -1 otherwise.

Built-in String Methods (Cont.)

- `index(str, beg = 0, end = len(string))`
  - Same as `find()`, but raises an exception if `str` not found.

Built-in String Methods (Cont.)

- `isalnum()`
  - Returns `true` if string has at least 1 character and all characters are alphanumeric and `false` otherwise.

Built-in String Methods (Cont.)

- `isalpha()`
  - Returns `true` if string has at least 1 character and all characters are alphabetic and `false` otherwise.

Built-in String Methods (Cont.)

- `isdigit()`
  - Returns `true` if string contains only digits and `false` otherwise.

Built-in String Methods (Cont.)

- `islower()`
  - Returns `true` if string has at least 1 cased character and all cased characters are in lowercase and `false` otherwise.

Built-in String Methods (Cont.)

- `isnumeric()`
  - Returns `true` if a unicode string contains only numeric characters and `false` otherwise.

Built-in String Methods (Cont.)

- `isspace()`
  - Returns `true` if string contains only whitespace characters and `false` otherwise.

Built-in String Methods (Cont.)

- `istitle()`
  - Returns `true` if string is properly "titlecased" and `false` otherwise.

Built-in String Methods (Cont.)

- `isupper()`
  - Returns `true` if string has at least one cased character and all cased characters are in uppercase and `false` otherwise.

Built-in String Methods (Cont.)

- `join(seq)`
  - Merges (concatenates) the string representations of elements in sequence `seq` into a string, with separator string.

Built-in String Methods (Cont.)

- `len(string)`
  - Returns the length of the `string`

Built-in String Methods (Cont.)

- `ljust(width[, fillchar])`
  - Returns a space-padded string with the original string left-justified to a total of `width` columns.

Built-in String Methods (Cont.)

- `lower()`
  - Converts all uppercase letters in string to lowercase.

Built-in String Methods (Cont.)

- `lstrip()`
  - Removes all leading whitespace in string.

Built-in String Methods (Cont.)

- `maketrans()`
  - Returns a translation table to be used in translate function.

Built-in String Methods (Cont.)

- `max(str)`
  - Returns the max alphabetical character from the string `str`.

Built-in String Methods (Cont.)

- `min(str)`
  - Returns the min alphabetical character from the string `str`.

Built-in String Methods (Cont.)

- `replace(old, new[, max])`
  - Replaces all occurrences of `old` in string with `new` or at most max occurrences if `max` given.

Built-in String Methods (Cont.)

- `rfind(str, beg = 0, end = len(string))`
  - Same as `index()`, but search backwards in string.

Built-in String Methods (Cont.)

- `rindex(str, beg = 0, end = len(string))`
  - Same as `index()`, but search backwards in string.

Built-in String Methods (Cont.)

- `rjust(width, [, fillchar])`
  - Returns a space-padded string with the original string right-justified to a total of `width` columns.

Built-in String Methods (Cont.)

- `rstrip()`
  - Removes all trailing whitespace of string.

Built-in String Methods (Cont.)

- `split(str = “”, num = string.count(str))`
  - Splits string according to delimiter `str` (space if not provided) and returns list of substrings; split into at most `num` substrings if given.

Built-in String Methods (Cont.)

- `splitlines(num = string.count('\n'))`
  - Splits string at all (or `num`) NEWLINEs and returns a list of each line with NEWLINEs removed.

Built-in String Methods (Cont.)

- `startswith(str, beg = 0, end = len(string))`
  - Determines if string or a substring of string (if starting index `beg` and ending index `end` are given) starts with substring `str`; returns `true` if so and `false` otherwise.

Built-in String Methods (Cont.)

- `strip([chars])`
  - Performs both `lstrip()` and `rstrip()` on string

Built-in String Methods (Cont.)

- `swapcase()`
  - Inverts case for all letters in string.

Built-in String Methods (Cont.)

- `title()`
  - Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.

Built-in String Methods (Cont.)

- `translate(table, deletechars = "")`
  - Translates string according to translation table `str(256 chars)`, removing those in the `del` string.

Built-in String Methods (Cont.)

- `upper()`
  - Converts lowercase letters in string to uppercase.

Built-in String Methods (Cont.)

- `zfill(width)`
  - Returns original string leftpadded with zeros to a total of `width` characters; intended for numbers, `zfill()` retains any sign given (less one zero).

Built-in String Methods (Cont.)

- `isdecimal()`
  - Returns `true` if a unicode string contains only decimal characters and `false` otherwise.

Strings - Lab

```
1 txt1 = '자바';txt2='파이썬'
2 num1= 5; num2=10
3 print('나는 %s보다 %s에 더 익숙합니다.' %(txt1, txt2))
4 print('%s은 %s보다 %d배 더 쉽습니다.' %(txt2, txt1, num1))
5 print('%d + %d = %d' %(num1, num2, num1+num2))
6 print('작년 세계 경제 성장률은 전년에 비해 %d%% 포인트 증가했다.' %num1)
7
```

Strings – Lab (Cont.)

```
1 listdata = [9.96, 1.27, 5.07, 6.45, 8.38, 9.29, 4.93, 7.73, 3.71, 0.93]
2 maxval = max(listdata)
3 minval = min(listdata)
4 print(maxval)     # 9.96이 출력됨
5 print(minval)     # 0.93이 출력됨
6
7 txt = 'Alotofthingsoccureachday'
8 maxval = max(txt)
9 minval = min(txt)
10 print(maxval);     # 'y'가 출력됨
11 print(minval)     # 'A'가 출력됨
12
13 maxval = max(2+3, 2*3, 2**3, 3**2)
14 minval = min('abz', 'a12')
15 print(maxval)    # 9가 출력됨
16 print(minval)     # 'abz'가 출력됨
17
```

Strings – Lab (Cont.)

```
1 txt = 'aAbBcCdDeEfFgGhHiIjJkK'
2 ret = txt[::2]
3 print(ret)          # 'abcdefghijk' 가 출력됨
4
```

```
1 txt = 'abcdefghijk'
2 ret = txt[::-1]
3 print(ret)          # 'kjihgfedcba' 가 출력됨
4
```

Strings – Lab (Cont.)

```
1 msg = input('임의의 문장을 입력하세요: ')
2 if 'a' in msg:
3     print('당신이 입력한 문장에는 a가 있습니다.')
4 else:
5     print('당신이 입력한 문장에는 a가 없습니다.')
6
```

```
1 msg = input('임의의 문장을 입력하세요: ')
2 if 'is' in msg:
3     print('당신이 입력한 문장에는 is가 있습니다.')
4 else:
5     print('당신이 입력한 문장에는 is가 없습니다.')
6
```

Strings – Lab (Cont.)

```
1 msg = input('임의의 문장을 입력하세요: ')
2 msglen = len(msg)
3 #msglen = len(msg.encode())
4 print('당신이 입력한 문장의 길이는 <%d> 입니다.' %msglen)
5
```

Strings – Lab (Cont.)

```
txt1 = 'A'
txt2 = '안녕'
txt3 = 'Warcraft Three'
txt4 = '3PO'
ret1 = txt1.isalpha()
ret2 = txt2.isalpha()
ret3 = txt3.isalpha()
ret4 = txt4.isalpha()
print(ret1)          # True가 출력됨
print(ret2)          # True가 출력됨
print(ret3)          # False가 출력됨
print(ret4)          # False가 출력됨

```

```
txt1 = '010-1234-5678'
txt2 = 'R2D2'
txt3 = '1212'
ret1 = txt1.isdigit()
ret2 = txt2.isdigit()
ret3 = txt3.isdigit()
print(ret1)          # False가 출력됨
print(ret2)          # False가 출력됨
print(ret3)          # True가 출력됨

```

Strings – Lab (Cont.)

```
1 txt1 = '안녕하세요?'
2 txt2 = '1. Title-제목을 넣으세요'
3 txt3 = '3피오R2D2'
4 ret1 = txt1.isalnum()
5 ret2 = txt2.isalnum ()
6 ret3 = txt3.isalnum ()
7 print(ret1)      # False가 출력됨
8 print(ret2)      # False가 출력됨
9 print(ret3)      # True가 출력됨
10
```

```
1 txt = '    양쪽에 공백이 있는 문자열입니다.    '
2 ret1 = txt.lstrip()
3 ret2 = txt.rstrip()
4 ret3 = txt.strip()
5 print('<'+txt+'>')
6 print('<'+ret1+'>')
7 print('<'+ret2+'>')
8 print('<'+ret3+'>')
9
```

Strings – Lab (Cont.)

```
1 txt = 'A lot of things occur each day, every day.'
2 word_count1 = txt.count('o')
3 word_count2 = txt.count('day')
4 word_count3 = txt.count(' ')
5 print(word_count1)  # 3이 출력됨
6 print(word_count2)  # 2가 출력됨
7 print(word_count3)  # 8이 출력됨
8
```

Strings – Lab (Cont.)

```
1 txt = 'A lot of things occur each day, every day.'
2 offset1 = txt.find('e')
3 offset2 = txt.find('day')
4 offset3 = txt.find('day', 30)
5 print(offset1)   # 22가 출력됨
6 print(offset2)   # 27이 출력됨
7 print(offset3)   # 38이 출력됨
8
```

Strings – Lab (Cont.)

```
1  url = 'http://www.naver.com/news/today=20160831'
2  log = 'name:홍길동 age:17 sex:남자 nation:조선'
3
4  ret1 = url.split('/')
5  print(ret1)
6
7  ret2 = log.split()
8  for data in ret2:
9      d1, d2 = data.split(':')
10     print('%s -> %s' %(d1, d2))
11
```

Strings – Lab (Cont.)

```
1 txt = 'My password is 1234'
2 ret1 = txt.replace('1', '0')
3 ret2 = txt.replace('1', 'python')
4 print(ret1)   # 'My Password is 0234'가 출력됨
5 print(ret2)   # 'My Password is python234'가 출력됨
6
7 txt = '매일 많은 일들이 일어납니다.'
8 ret3 = txt.replace('매일', '항상')
9 ret4 = ret3.replace('일', '사건')
10 print(ret3)   # '항상 많은 일들이 일어납니다.'가 출력됨
11 print(ret4)   # '항상 많은 사건들이 사건어납니다.'가 출력됨
12
```

Strings – Lab (Cont.)

```
1 u_txt = 'I love python'
2 b_txt = u_txt.encode()
3 print(u_txt)
4 print(b_txt)
5
6 ret1 = 'I' == u_txt[0]
7 ret2 = 'I' == b_txt[0]
8 print(ret1)    # True가 출력됨
9 print(ret2)    # False가 출력됨
10
```

Strings – Lab (Cont.)

```
1 b_txt = b'A lot of things occur each day.'
2 u_txt = b_txt.decode()
3 print(u_txt)
4
```

Strings – Lab (Cont.)

```
1 strdata = input('정렬할 문자열을 입력하세요: ')
2 ret1 = sorted(strdata)
3 ret2 = sorted(strdata, reverse=True)
4 print(ret1)
5 print(ret2)
6 ret1 = ''.join(ret1)
7 ret2 = ''.join(ret2)
8 print('오름차순으로 정렬된 문자열은 <' + ret1 + '> 입니다.')
9 print('내림차순으로 정렬된 문자열은 <' + ret2 + '> 입니다.')
10
```

Strings – Lab (Cont.)

```
1 expr1 = '2+3'
2 expr2 = 'round(3.7)'
3 ret1 = eval(expr1)
4 ret2 = eval(expr2)
5 print('<%s>를 eval()로 실행한 결과: ' %expr1, end=' '); print(ret1)
6 print('<%s>를 eval()로 실행한 결과: ' %expr2, end=' '); print(ret2)
7
```

Strings – Lab (Cont.)

```
1  num = input('아무 숫자를 입력하세요: ')
2
3  if num.isdigit():
4      num = num[::-1]
5      ret = ''
6      for i, c in enumerate(num):
7          i += 1
8          if i != len(num) and i%3 == 0:
9              ret += (c + ',')
10         else:
11             ret += c
12     ret = ret[::-1]
13     print(ret)
14 else:
15     print('입력한 내용 [%s]: 숫자가 아닙니다.' %num)
16
```

Strings – Lab (Cont.)

```
1  text = input('문장을 입력하세요: ')
2
3  ret = ''
4  for i in range(len(text)):
5      if i != len(text)-1:
6          ret += text[i+1]
7      else:
8          ret += text[0]
9
10 print(ret)
11
```

Strings – Lab (Cont.)

- 홍길동씨의 주민등록번호는 881120-1068234이다. 홍길동씨의 주민등록번호를 연월일(yyyymmdd)부분과 그 뒤의 숫자부분으로 나누어 출력해 보자. (ex. Slice operator 사용하기)