Open Source Programming

Lecture-05
Python

Contents

- Python Language
- Types / Numbers
- Operations (Arithmetic / Comparison / Boolean)
- Control (if, for, while, …)
- String Formatting

Python Language

- **Python**
  - High-level, general-purpose programming language
  - Created by Guido van Rossum (1991)
  - A global community of programmers develops and maintains Python, an open source reference implementation

  - Code readability
  - Object-oriented language
  - Dynamically typing

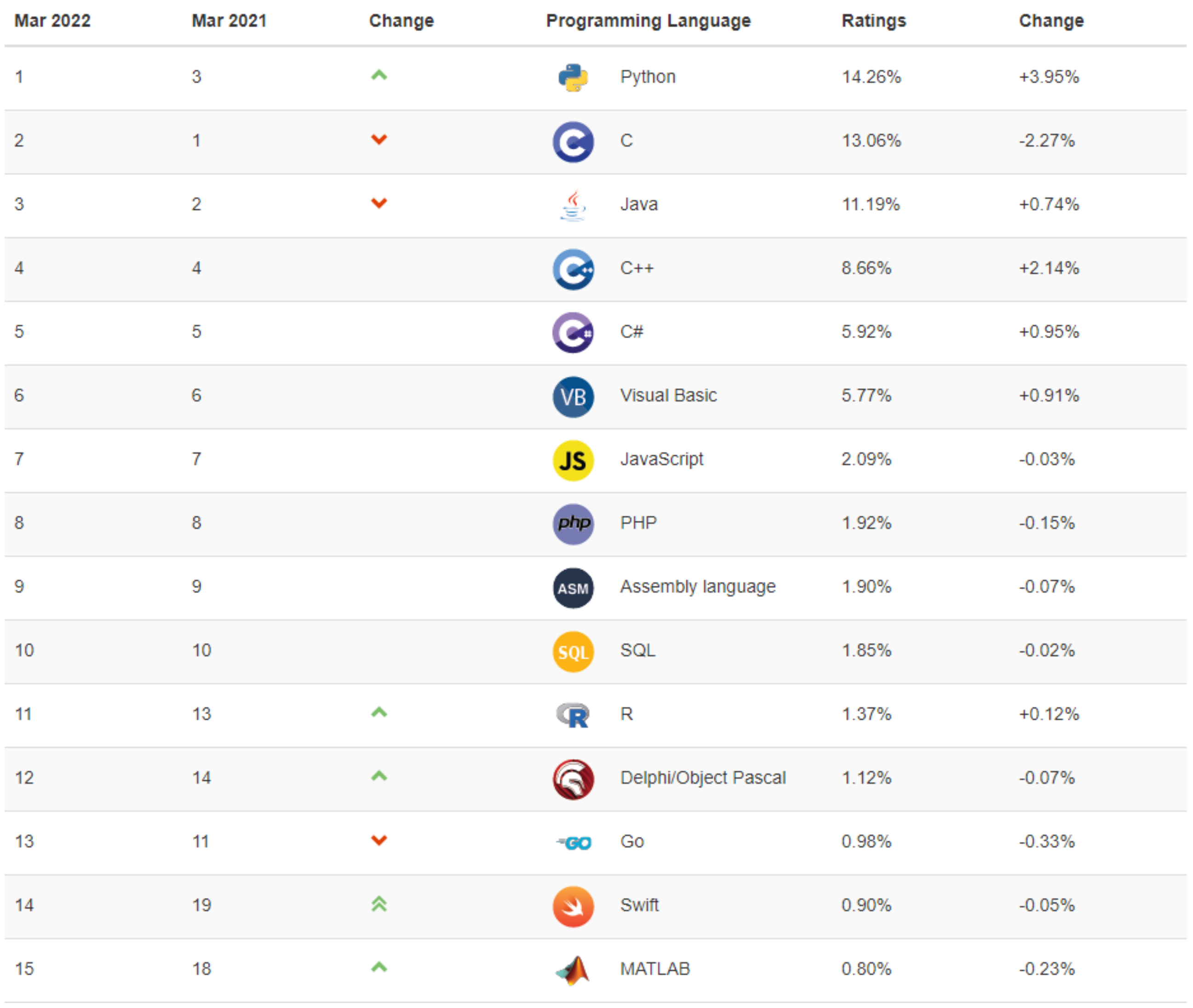
Gudi Van Rossum

- **Interpreter**
  - Line-by-line processing of code at runtime
  - Available for many OS (windows, linux, etc.)

Python Language

https://www.tiobe.com/tiobe-index/

- **TIOBE Index for March 2022**

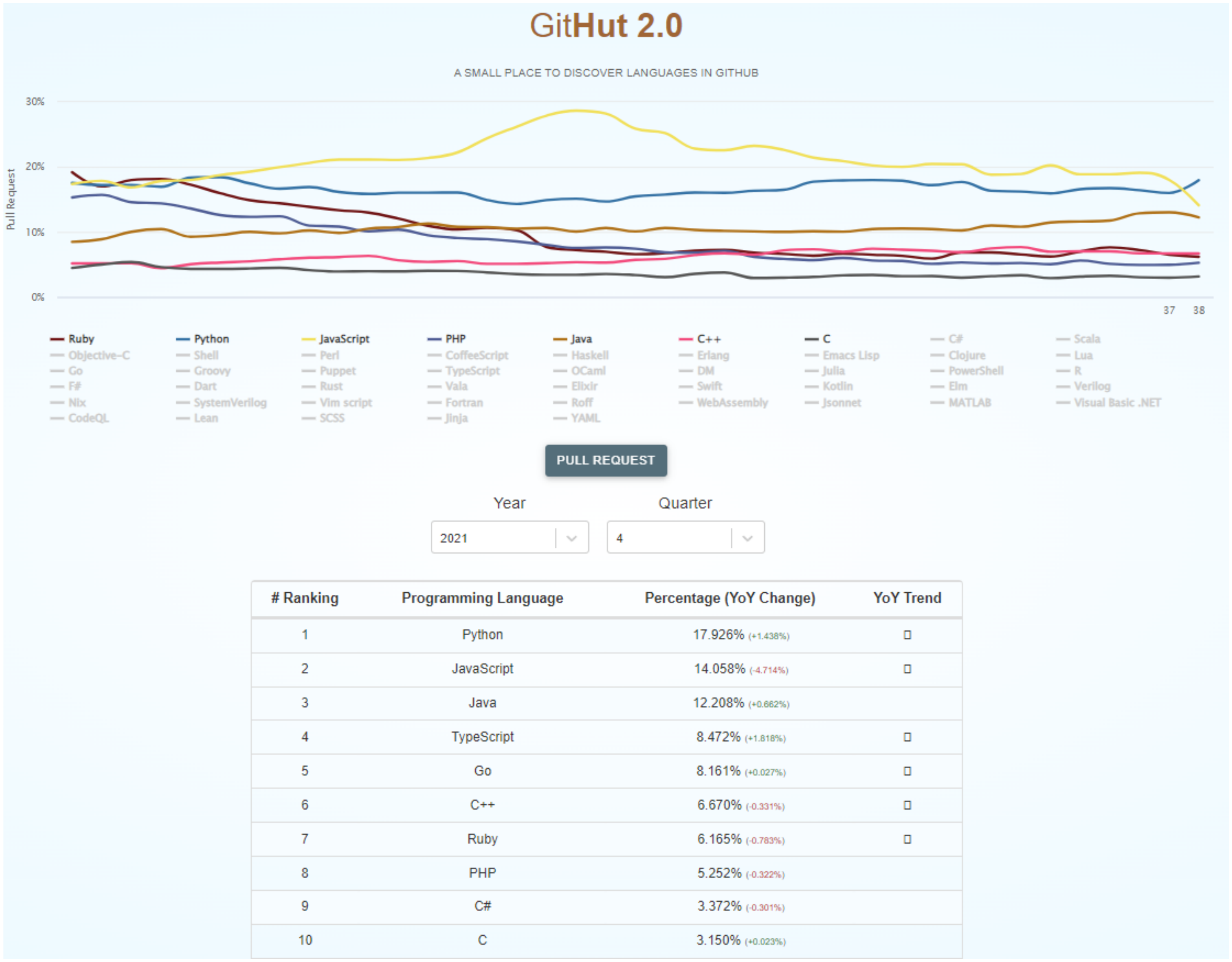


Mar 2022	Mar 2021	Change	Programming Language	Ratings	Change
1	3	▲	Python	14.26%	+3.95%
2	1	▼	C	13.06%	-2.27%
3	2	▼	Java	11.19%	+0.74%
4	4		C++	8.66%	+2.14%
5	5		C#	5.92%	+0.95%
6	6		Visual Basic	5.77%	+0.91%
7	7		JavaScript	2.09%	-0.03%
8	8		PHP	1.92%	-0.15%
9	9		Assembly language	1.90%	-0.07%
10	10		SQL	1.85%	-0.02%
11	13	▲	R	1.37%	+0.12%
12	14	▲	Delphi/Object Pascal	1.12%	-0.07%
13	11	▼	Go	0.98%	-0.33%
14	19	▲▲	Swift	0.90%	-0.05%
15	18	▲	MATLAB	0.80%	-0.23%

Python Language

https://madnight.github.io/githut/#/pull_requests/2020/4

- **Issues in GitHub (2021 - 4 Quarter)**



# Ranking	Programming Language	Percentage (YoY Change)	YoY Trend
1	Python	17.926% (+1.438%)	□
2	JavaScript	14.058% (-4.714%)	□
3	Java	12.208% (+0.662%)	
4	TypeScript	8.472% (+1.818%)	□
5	Go	8.161% (+0.027%)	□
6	C++	6.670% (-0.331%)	□
7	Ruby	6.165% (-0.783%)	□
8	PHP	5.252% (-0.322%)	
9	C#	3.372% (-0.301%)	
10	C	3.150% (+0.023%)	

Python Language

- **Python Version in Ubuntu 20.04**

```
$ python --version
Python 2.7.18

$ ls /usr/bin/ | grep python
...
python2
python2.7
...
python3
python3.8
...
```

Python Language

- Config Python Version : 2.7 → 3.8

```
$ sudo update-alternatives --install /usr/bin/python python
/usr/bin/python2.7 1
$ sudo update-alternatives --install /usr/bin/python python
/usr/bin/python3.8 2

$ sudo update-alternatives --config python
 There are 2 choices for the alternative python (providing
/usr/bin/python).

  Selection    Path              Priority   Status
------------------------------------------------------------
* 0            /usr/bin/python3.8   2         auto mode
  1            /usr/bin/python2.7   1         manual mode
  2            /usr/bin/python3.8   2         manual mode

Press <enter> to keep the current choice[*], or type selection number:
2
$ python --version
Python 3.8.10
```

Python Language

- **Running Python in Interpreter (Shell Mode)**

```
$ python
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> 3+4
7
>>> w=“hello”
>>> w
‘hello’
>>> print(w)
hello
>>> w[2]
‘l’
>>>
```

  - Use `exit()` or `Ctrl-D(EOF)` to exit

Python Language

- **Running in Script Mode using *.py files**

```
$ vi myprog.py
```

```
if __name__==‘__main__’:
        word = ‘hello’
        print(word)
```

_ _

```
$ python myprog.py
hello
$
```

Script Mode

- **Try running a script**

  (1) Create a text file "myprog.py" with this content.

```
import sys

if __name__==‘__main__’:
       print(sys.argv)
       word = ‘hello’
       print(word)
```

  (2) Type "**python ./myprog.py myparam**" to run

```
$ python ./myprog.py myparam
['myprog.py', `myparam']
hello
$
```

Script Mode

- **Executable python script**

  (1) Add one line at the top

```
#!/usr/bin/python
import sys

if __name__==‘__main__’:
       print(sys.argv)
       word = ‘hello’
       print(word)
```

  (2) chmod 755 myprog.py

  (3) Type "`./myprog.py myparam`" to run

```
$ chmod 755 myprog.py
$ ./myprog.py myparam
['myprog.py', `myparam']
hello
$
```

Indentation

- *No brackets to indicate blocks !!!*
- **Indentation matters (tab key)**

```
if True:
        print "True"
else:
        print "False"
```

- ***Space → Error !!***

```
if True:
 print "True"
else:
 print "False"
```

Quotation & Comments

- word = `'word'`
- sentence = `"This is a sentence."`
- paragraph = `"""This is a paragraph. It is made up of multiple lines and sentences."""`

```
#!/usr/bin/python

“”””

Python program
This is my first…
“”””


# First comment
print("Hello, Python!“) # second comment
```

Types in Python

- **Numeric Types**
  - int : 42- *may be transparently expanded to long through* 438324932L
  - float : 2.171892
  - complex : 4 + 3j
- **Sequence Types**
  - range : range(1, 10, 1)
  - list : [69, 6.9, ‘mystring’, True]
  - tuple : (69, 6.9, ‘mystring’, True) *immutable*
- **Text Sequence Type**
  - string : ‘MyString’, u‘MyString’ (unicode)
- **Set Type**
  - set/frozenset : set([69, 6.9, ‘str’, True]) - *no duplicates & unordered*
    frozenset([69, 6.9, ‘str’, True]) - *immutable*
- **Mapping Type**
  - dictionary : {‘key 1’: 6.9, ‘key2’: False} - *group of key and value pairs*

Numbers in Python

- **Integer**
- **Long**
  - 2L**100 → 1267650600228229401496703205376L
    - x ** y or pow(x, y) : *x to the power y*
- **Float**
  - 3.14, 0.5
- **Hexa**
  - 0xF3AA
- **Complex**
  - (-1+0j)
  - Appending 'j' or 'J' to a numeric literal yields an imaginary number (a complex number with a zero real part)

Arithmetic Operations

- The operators + - * / % ** ( ) all work for real numbers.

```
$ python
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> 2+4*3
14
>>> 2.5*3
7.5
>>> 2.5*2
5.0
>>> 35/7
5.0
>>> 35//7
5
>>> 5%2
1
>>> 5%2.0
1.0
```

Comparison Operations

- **8 comparison operations in Python**
- **same priority**
- **chained arbitrarily**
  - ex) `x < y <= z`
    `x < y and y <= z`

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

```
>>> x = 1
>>> y = 3
>>> z = 5
>>> x < y <= z
True
>>> x > y <= z
False
>>>
```

Comparison Operations

- **`is`** : test for object identify
  - true if and only if x and y are the same object

```
>>> a = 5
>>> b = 5.0
>>> a == b
True
>>> a is b
False
>>>
>>> id(a)
10914624
>>> id(b)
140222025350384
```

- **`id`** : return the “identity” of an object
  - an integer address which is guaranteed to be unique and constant for this object during its lifetime

Comparison Operations

```
>>> a = 5
>>> b = 5
>>> a == b
True
>>> a is b
True
>>>
>>> id(a)
10914624
>>> id(b)
10914624
```

- `id`: return the "identity" of an object
  - Two objects with non-overlapping lifetimes may have the same id() value.

Boolean Operations

- **and, or, not**

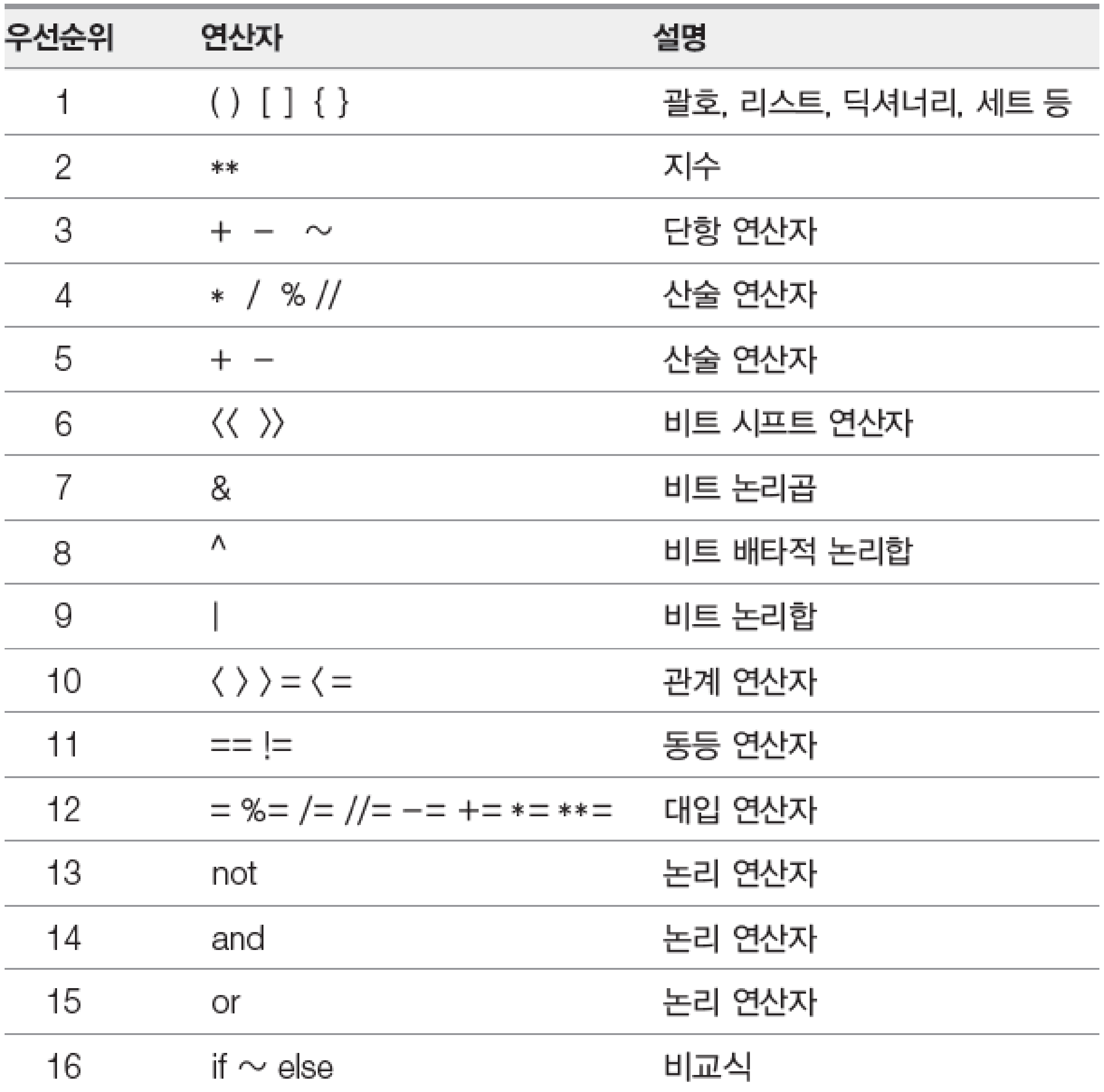
```
>>> a = 99
>>> (a<100)and(a<200)
True
>>> (a<90)and(a<100)
False
>>> (a>90)and(a<100)
True
>>> 90<a<100
True
>>>
>>> (a<90)or(a<100)
True
>>> not(a==100)and(a<200)
True
>>> not(a==99)and(a<200)
False
>>>
```

Boolean + Comparison Operations

- 1) Comparisons (is, is not, ==, !=, <, >, <=, >=)
- 2) Boolean (not, and, or)

```
>>> 10==10 and 10!=5
True
>>> 10>5 and 10<3
False
>>> not 10>5
False
>>> not 1 is 1.0
True
>>>
```

Priority of Operations



우선순위	연산자	설명
1	( ) [ ] { }	괄호, 리스트, 딕셔너리, 세트 등
2	**	지수
3	+ − ~	단항 연산자
4	* / % //	산술 연산자
5	+ −	산술 연산자
6	〈〈 〉〉	비트 시프트 연산자
7	&	비트 논리곱
8	^	비트 배타적 논리합
9	|	비트 논리합
10	〈 〉 〉= 〈=	관계 연산자
11	== !=	동등 연산자
12	= %= /= //= −= += *= **=	대입 연산자
13	not	논리 연산자
14	and	논리 연산자
15	or	논리 연산자
16	if ~ else	비교식

Conditionals : if

```
if (value is not None) and (value == 1):
       print(“value equals 1”)

if (list1 <= list2) and (not age < 80):
       print(“1 = 1, 2 = 2, but 3 <= 7 so its True”)

if (score >= 90):
       print(“A”):
elif (score >=80):
       print(“B”):
else:
       print(“C”):

if (job == "millionaire") or (state != "dead"):
       print(“a suitable husband found”)
else:
       print(“not suitable”)

if ok: print(“ok”)
```

While Loop

```
while <condition>:
       <expression>
       <expression>
       ...
```

- <condition> evaluates to a Boolean
- If <condition> is True,
  do all the steps inside the while code block
- Check <condition> again
- Repeat until <condition> is False

```
#!/usr/bin/python

n = input("You're in the Lost Forest. Go left or right? ")
while n == "right":
    n = input("You're in the Lost Forest. Go left or right? ")
    print("You got out of the Lost Forest!")
```

For Loop

```
for <variable> in range(<some_num>):
       <expression>
       <expression>
       ...
```

- Each time through the loop, <variable> takes a value
- First time, <variable> starts at the smallest value
- Next time, <variable> gets the prev value + 1

```
#!/usr/bin/python

for letter in 'Python':     # First Example
   print(“Current Letter :”, letter)

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # Second Example
   print(“Current fruit :”, fruit)
```

Range

```
range(start, stop, step)
```

- default values are start = 0 and step = 1 and optional
- loop until value is stop - 1

```
>>> list(range(0,10,1))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(0,10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> list(range(0,10,2))
[0, 2, 4, 6, 8]
>>> list(range(10,0,-1))
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> list(range(10,0,-2))
[10, 8, 6, 4, 2]
```

Range

```
range(start, stop, step)
```

- default values are start = 0 and step = 1 and optional
- loop until value is stop - 1

```
#!/usr/bin/python

mysum = 0
for i in range(7, 10):
        mysum += i
        print(mysum)
```

```
#!/usr/bin/python

mysum = 0
for i in range(5, 11, 2):
        mysum += i
        print(mysum)
```

break

```
while <condition_1>:
        while <condition_2>:
                if <expression_a>:
                        break
                <expression_b>
        <expression_c>
```

- immediately exits whatever loop it is in
- skips remaining expressions in code block
- exits only innermost loop!

```
#!/usr/bin/python

mysum = 0
for i in range(5, 12, 2):
        mysum += i
        if (mysum > 20):
                break
        print(i)
print("sum range:", mysum)
```

continue

```
while <condition_1>:
        if <expression_a>:
                continue
        <expression_b>
<expression_c>
```

```
#!/usr/bin/python

mysum = 0
for i in range(1, 101):
        if (i%3 == 0): # excluding multiples of 3
                continue
        mysum += i
print("sum range:", mysum)
```

String Type

- immutable sequences of unicode code points
- String literals are written in a variety of ways
  - Triple quoted strings may span multiple lines

```
>>> print('allows embedded "double" quotes')
allows embedded "double" quotes
>>> print("allows embedded 'single'quotes")
allows embedded 'single'quotes
>>> print('''three single quotes''')
three single quotes
>>> print("""three double quotes""")
three double quotes
>>>
>>> str1="hello"
>>> str2="world"
>>> print(str1,str2)
hello world
>>> type(str1)
<class 'str'>
>>> str1[3]
'l'
```

Print and Parameters

- print with parameters : `sep`, `end`

```
>>> print(1,2,3)
1 2 3
>>> print("hello","world")
hello world
>>>
>>> print(1,2,3,sep=',')
1,2,3
>>> print("hello","world",sep="")
helloworld
>>> print(1024,768,sep='x')
1024x768
>>>
>>> print(1); print(2); print(3)
1
2
3
>>> print(1,end=""); print(2,end=""); print(3)
123
```

String Formatting

- **Formatting operator %**

```
>>> print("my name is %s and weight is %d kg!" % ('Tom',60))
my name is Tom and weight is 60 kg!
```

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%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

```
>>> print("%d" % 123)
123
>>> print("%5d" % 123)
  123
>>> print("%05d" % 123)
00123
>>>
>>> print("%f" % 123.45)
123.450000
>>> print("%7.1f" % 123.45)
  123.5
>>> print("%7.3f" % 123.45)
123.450
>>>
>>> print("%s" % "hello")
hello
>>> print("%10s" % "hello")
     hello
```

Input

- read a line from input, converts it to a string, and return

```
#!/usr/bin/python

strname = input("input your name:")
print("my name is %s" % strname)

weight = int(input("input your weight:"))
print("my weight is %d" % weight)
```

String Formatting Operation

- The string on which this method is called can contain literal text or replacement fields delimited by braces {}
- Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument
- Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument

```
>>> print("the sum of 1+2 is {0}".format(3))
the sum of 1+2 is 3
>>> print("the sum of 1+2 is {0}".format(1+2))
the sum of 1+2 is 3
```

String Formatting Operation

```
>>>
>>> print("the sum of {0}+{1} is {2}".format(1,2,3))
the sum of 1+2 is 3
>>>
>>> print("the sum of {}+{} is {}".format(1,2,3))
the sum of 1+2 is 3
>>>
>>> print("the sum of {0}+{1} is {num}".format(1,2,num=3))
the sum of 1+2 is 3
>>>
>>> print("the sum of {0:d}+{1:d} is {2:05d}".format(1,2,3))
the sum of 1+2 is 00003
>>>
>>> print("{0}+{1}={2}, {0}-{1}={3}".format(1,2,1+2,1-2))
1+2=3, 1-2=-1
>>>
>>> print("{1}+{0}={2}, {1}-{0}={3}".format(1,2,2+1,2-1))
2+1=3, 2-1=1
>>>
```

String Formatting Operation

```
>>> print('{:>5}'.format('123'))
  123
>>> print('{:>05}'.format('123'))
00123
>>> print('{:>010}'.format('123'))
0000000123
>>> print('{:0>10}'.format('123'))
0000000123
>>> print('{:x>10}'.format('123'))
xxxxxxx123
>>>
>>> print('{:>10}'.format('hello'))
     hello
>>>
>>> print('{:.3f}'.format(12.3456789))
12.346
```

Built-in Functions

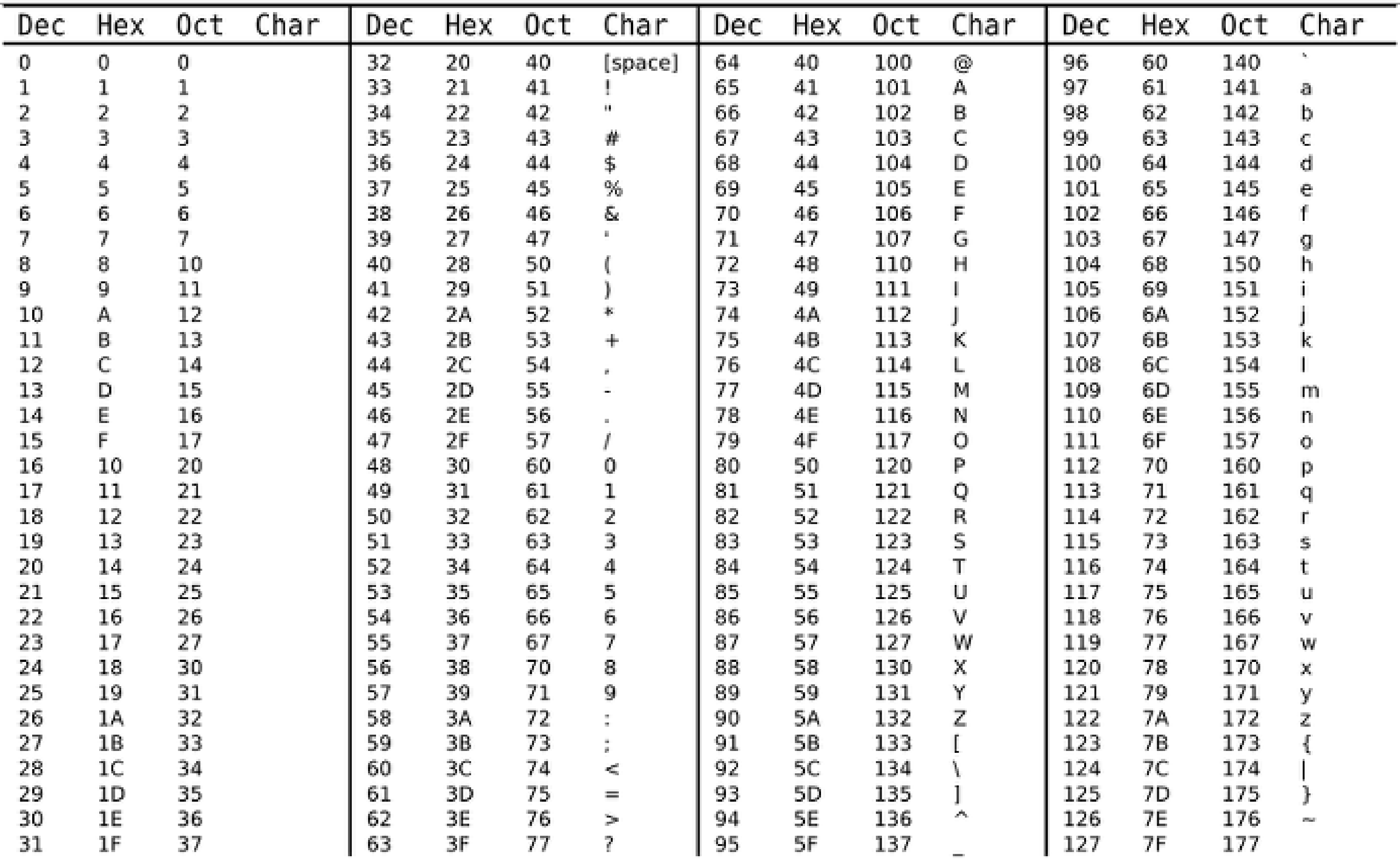
https://docs.python.org/3.8/library/functions.html

- `len(`***`string`***`)`
  - number of characters in a string (including spaces)
  - Example:

```
name = "Martin Douglas Stepp‟
length = len(name)
```

- **Characters map to numbers using standardized mappings such as *ASCII* and *Unicode*.**
- `ord(`***`text`***`)`
  - converts a string into a number
  - Example: `ord("a")` is 97, `ord("b")` is 98, ...
- `chr(`***`number`***`)`
  - converts a number into a string
  - Example: `chr(99)` is "c"

ASCII Table



Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Contents

List

- **Properties**
  - Ordered collections of arbitrary objects
  - Mutable
  - Accessed by offset
  - Arrays of object references
- **Basic operations**

```
>>> len([1, 2, 3])                  # Length
3
>>> [1, 2, 3] + [4, 5, 6]           # Concatenation (+)
[1, 2, 3, 4, 5, 6]
>>> ['Ni!'] * 4                     # Repetition (*)
['Ni!', 'Ni!', 'Ni!', 'Ni!']
>>> str([1, 2]) + "34"
'[1, 2]34'
>>> [1, 2] + "34"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "str") to list
```

List Iteration and Comprehensions

```
>>> 3 in [1,2,3]
True
>>> for x in [1, 2, 3]:
...             print(x)
...
1
2
3
```

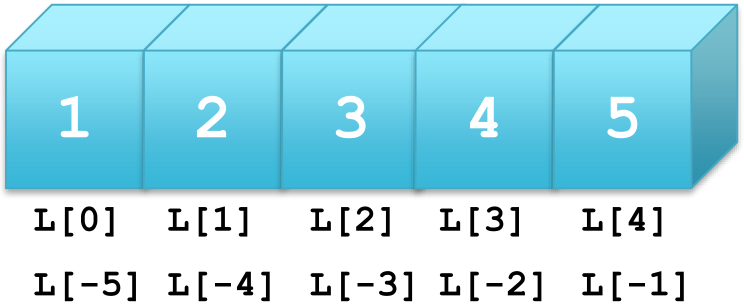
```
>>> res = []
>>> for c in 'SPAM':          # List comprehension equivalent
...     res.append(c * 4)
...
>>> res
['SSSS', 'PPPP', 'AAAA', 'MMMM']


>>> res = [c * 4 for c in 'SPAM'] # List comprehensions
>>> res
['SSSS', 'PPPP', 'AAAA', 'MMMM']
```

List Update

- **List is mutable**
  - Can modify the list objects (overwrite)

```
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[1] = 'eggs'    # Index assignment
>>> L
['spam', 'eggs', 'SPAM!']
```

- **List index**



```
>>> L = [1, 2, 3, 4, 5]
>>> L[1]
2
>>> L[-1]
5
>>> L[-2]
4
```

List Slice

- **list [** *start : end+1 : (step)* **]**
  - Slice of list from start to end
  - Slice of list from start to end with step

```
>>> L = [1, 2, 3, 4, 5]
>>> L[0:3]
[1, 2, 3]
>>> L[:2]
[1, 2]
>>> L[2:]
[3, 4, 5]
>>> L[:]
[1, 2, 3, 4, 5]
>>> L[0:5:2]
[1, 3, 5]
>>> L[:5:2]
[1, 3, 5]
>>> L[::2]
[1, 3, 5]
```

Mutable Assignment

```
>>> L = [1, 2, 3, 4, 5]
>>> L[1] = 2.5                 # Replacement
>>> L
[1, 2.5, 3, 4, 5]
>>> L[2:4] = [3.5, 4.5]        # Replacement
>>> L
[1, 2.5, 3.5, 4.5, 5]
>>> L[2:3] = [3.7, 3.9]        # Replacement/insertion
>>> L
[1, 2.5, 3.7, 3.9, 4.5, 5]
>>> L[2:2] = [3.5, 3.6]        # Insertion (replace nothing)
>>> L
[1, 2.5, 3.5, 3.6, 3.7, 3.9, 4.5, 5]
>>> L[2:3] = []                # Deletion (insert nothing)
>>> L
[1, 2.5, 3.6, 3.7, 3.9, 4.5, 5]
>>> del L[3:5]
>>> L
[1, 2.5, 3.6, 4.5, 5]
>>>
```

List Methods

- **Built-in functions**
  - `len(`*`s`*`)` : length of *s*
  - `min(`*`s`*`)` : smallest item of *s*
  - `max(`*`s`*`)` : largest item of *s*
  - `sum(`*`s`*`)` : sums items of *s* and returns the total

- ***s***.count(***x***)
  - total number of occurrences of ***x*** in ***s***

- ***s***.index(***x***[, i[, j]])
  - index of the first occurrence of ***x*** in ***s*** (at or after index i and before index j)

List Methods

- ***s***.append(***x***)
  - appends ***x*** to the end of the sequence (same as ***s***[len(***s***):len(***s***)] = [***x***])
- ***s***.clear()
  - removes all items from ***s*** (same as del ***s***[:])
- ***s***.copy()
  - creates a shallow copy of ***s*** (same as ***s***[:])
- ***s***.extend(***t***) or ***s***+= ***t***
  - extends ***s*** with the contents of ***t*** (same as ***s***[len(***s***):len(***s***)]=***t***)

List Methods

- **append vs. *s*+[*x*]**
  - append : in-place update
  - + : create new list

- **extend vs. append**
  - extend: iterates through and adds each item in an iterable object
  - append: simply adds a single item as is without iterating through it

```
>>> L=[1,2]
>>> L.extend([3,4,5])
>>> L
[1, 2, 3, 4, 5]
>>> L.append([6,7,8])
>>> L
[1, 2, 3, 4, 5, [6, 7, 8]]
```

List Methods

- ***s***.insert(i, ***x***)
  - insert ***x*** into ***s*** at the index given by i (same as ***s***[i:i] = [***x***])
- ***s***.pop([i])
  - retrieves the item at i and also removes it from ***s***
- ***s***.remove(***x***)
  - remove the first item from ***s*** where ***s***[i] == ***x***
- ***s***.reverse()
  - reverses the items of ***s*** in place
- ***s***.sort()
  - sorts the list in place using only < comparisons between items

List Methods

```
>>> L
[1, 2, 3, 4, 5, [6, 7, 8]]
>>> L.pop()   # Delete and return last item (by default: -1)
[6, 7, 8]
>>> L
[1, 2, 3, 4, 5]
>>> L.reverse()      # In-place reversal method
>>> L
[5, 4, 3, 2, 1]
>>>
>>>
>>> s = ['eat', 'more', 'SPAM!']
>>> s.append('please') # Append method call: add item at end
>>> s
['eat', 'more', 'SPAM!', 'please']
>>> s.sort()              # Sort list items ('S' < 'e')
>>> s
['SPAM!', 'eat', 'more', 'please']
>>>
```

List Methods

```
>>> s = ['spam', 'eggs', 'ham']
>>> s.index('eggs')          # Index of an object (search/find)
1
>>> s.insert(1, 'toast')     # Insert at position
>>> s
['spam', 'toast', 'eggs', 'ham']
>>> s.remove('eggs')         # Delete by value
>>> s
['spam', 'toast', 'ham']
>>> s.pop(1)                 # Delete by position
'toast'
>>> s
['spam', 'ham']
>>>
>>> del s[0]                 # Delete one item
>>> s
['ham']
>>>
```

List Sorting

- **Default order: ascending**
  - 'reverse' argument can be used to specify descending order

```
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort()                    # Sort with mixed case
>>> L
['ABD', 'aBe', 'abc']
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort(key=str.lower)   # Normalize to lowercase
>>> L
['abc', 'ABD', 'aBe']
>>>
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort(key=str.lower, reverse=True)  # Change sort order
>>> L
['aBe', 'ABD', 'abc']
```

List Sorting

- **append and sort do not return a value**
  - L=L.append(X) is wrong
- In order to build new sorted list, use **sorted** built-in function

```
>>> L = ['abc', 'ABD', 'aBe']
>>> sorted(L, key=str.lower, reverse=True) # Sorting built-in
['aBe', 'ABD', 'abc']
```

- Converting to lower case before sorting

```
>>> L = ['abc', 'ABD', 'aBe']
>>> sorted([x.lower() for x in L], reverse=True)
# Pretransform items: differs!
['abe', 'abd', 'abc']
```

String Access

- Use the square brackets for slicing along with the index or indices to obtain substring

```
>>> S = ‘Spam’        # 4-character string
>>> S[1:3]    # Slice of S from offsets 1 through 2 (not 3)
'pa'
>>> S[1:]     # Everything past the first (1:len(S))
'pam'
>>> S         # S itself hasn't changed
'Spam'
>>> S[0:3]    # Everything but the last
'Spa'
>>> S[:3]     # Same as S[0:3]
'Spa'
>>> S[:-1]   # Everything but the last again, but simpler (0:-1)
'Spa'
>>> S[:]      # All of S as a top-level copy (0:len(S))
'Spam'
```

Immutability of String

- *Immutable*: cannot be changed

```
>>> S
'Spam'
>>> S[0] = 'z'      # Immutable objects cannot be changed
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> S = 'z' + S[1:]
# But we can run expressions to make new objects
>>> S
'zpam`
>>>
>>> a = ‘hello’
>>> b = ‘world’
>>> a+b
‘helloworld’
>>> a*3
‘hellohellohello’
>>>
```

Immutability of String

- In-place update of characters in string

```
>>> S = 'shrubbery'
>>> L = list(S)          # Expand to a list: [...]
>>> L
['s', 'h', 'r', 'u', 'b', 'b', 'e', 'r', 'y']
>>> L[1] = 'c'           # Change it in place
>>> L
['s', 'h', `c', 'u', 'b', 'b', 'e', 'r', 'y']
>>>
>>> ''.join(L)           # Join with empty delimiter
'scrubbery'
>>> '-'.join(L)          # Join with dash
's-c-r-u-b-b-e-r-y'
>>>
```

String Methods

- ***str***.lower()
  - lowercase version of a string
- ***str***.upper()
  - uppercase version of a string
- ***str***.swapcase()
  - string with uppercase characters converted to lowercase and vice versa
- ***str***.title()
  - titlecased version of the string where words start with an uppercase character and the remaining characters are lowercase.

String Methods

- ***str***.count(text [, start[, end]])
  - Return the number of non-overlapping occurrences of substring text in the range [start:end]
- ***str***.find(text[, start[, end]])
  - Return the lowest index in the string where substring text is found within the slice s[start:end]
- ***str***.rfind(text [, start[, end]])
  - Return the highest index in the string where substring text is found, such that sub is contained within [start:end]
- ***str***.index(text [, start[, end]])
  - Like find(), but raise *ValueError* when the substring is not found
- ***str***.join(list)
  - Return a string which is the concatenation of the strings in list

String Methods

- ***str*.isdigit()**
  - Return true if all characters in the string are digits and there is at least one character, false otherwise

- ***str*.isalpha()**
  - Return true if all characters in the string are alphabetic and there is at least one character, false otherwise

- ***str*.isalnum()**
  - Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise

String Methods

- ***str*.split(*sep=None*)**
  - Return a list of the words in the string, using *sep* as the delimiter string
- ***str*.splitlines()**
  - Return a list of the lines in the string, breaking at line boundaries
- ***str*.strip([t])**
  - Return a copy of the string with the leading and trailing characters removed
- ***str*.rstrip([t])**
  - Return a copy of the string with trailing characters removed

```
>>> '   spacious   '.strip()
'spacious'
>>> 'www.example.com'.strip('cmowz.')
'example'
>>> '   spacious   '.rstrip()
'   spacious'
>>> 'mississippi'.rstrip('ipz')
'mississ'
```

String Methods

```
>>> S = 'Spam'
>>> S.find('pa')     # Find the offset of a substring in S
1
>>> S.replace('pa', 'XYZ')
# Replace occurrences of a string in S with another
'SXYZm'
>>> S
'Spam'
>>> line = 'aaa,bbb,ccccc,dd'
>>> line.split(',')
# Split on a delimiter into a list of substrings
['aaa', 'bbb', 'ccccc', 'dd']
>>> S = 'spam'
>>> S.isalpha()
# Content tests: isalpha, isdigit, etc.
True
>>> line = 'aaa,bbb,ccccc,dd\n'
>>> line.rstrip()
# Remove whitespace characters on the right side
'aaa,bbb,ccccc,dd'
```

String Methods

```
>>> ss = "hello, Open Source Software is powered!"
>>> ss.split(',')
['hello', ' Open Source Software is powered!']
>>> ss.split()
['hello,', 'Open', 'Source', 'Software', 'is', 'powered!']
>>>
>>> ss_num = "one:two:three"
>>> ss_num.split(':')
['one', 'two', 'three']
>>>
>>> ss_nl = "one\ntwo\nthree"
>>> ss_nl.splitlines()
['one', 'two', 'three']
>>>
>>> a,b = input("two numbers:").split()
two numbers:100 200
>>> print(a,b)
100 200
>>>
```

String Formatting

- **Number formatting**

```
>>> '{0:.2f}'.format(1.2345)       # String method
'1.23'
>>> format(1.2345, '.2f')          # Built-in function
'1.23'
>>> '%.2f' % 1.2345                # Expression
'1.23'
>>>
>>>
>>> '{:,.2f}'.format(296999.2567) # Separators, decimal digits
'296,999.26'
>>> '%.2f | %+-05d' % (3.14159, -42) # Digits, padding, signs
'3.14 | -0042'
>>> '{:10} = {:10}'.format('spam', 123.4567)
'spam       =   123.4567'
>>> '{:>10} = {:<10}'.format('spam', 123.4567)
'      spam = 123.4567  '
>>>
```

Tuple

- **Tuples are immutable sequences**
- used to store collections of heterogeneous data

```
>>> week1 = (‘sun’,’mon’,’tue’,’wed’,’thu’,’fri’,’sat’)
>>> week2 = ‘SUN’,’MON’,’TUE’,’WED’,’THU’,’FRI’,’SAT’
>>> week1
(‘sun’,’mon’,’tue’,’wed’,’thu’,’fri’,’sat’)
>>> week2
(‘SUN’,’MON’,’TUE’,’WED’,’THU’,’FRI’,’SAT’)
>>> week1[1] = ‘MON’
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> week1.append(‘test’)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'apped'
>>> del(week1[0])
...
>>> del(week1)
```

Tuple

```
>>> tt = (1,2,3,4,5)
>>> (n1,n2,n3,n4,n5) = tt
>>> n3
3
>>> (a1,a2) = tt
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
>>>
>>> a,b = 1,2          # tuple assignment (a,b) = (1,2)
>>> a
1
>>> b
2
>>>
```

Dictionary

- **Properties**
  - *unordered* collection of objects and *mutable*
  - key-value pair **{key:value}**
  - ***Accessed by key***, <u>*not by offset or index*</u>
  - Variable length, heterogeneous
  - Arbitrarily nestable

- **Basic dictionary operations**

```
>>> D = {'spam': 2, 'ham': 1, 'eggs': 3} # Make a dictionary
>>> D['spam']                           # Fetch a value by key
2
>>> D
{'eggs': 3, 'spam': 2, 'ham': 1}
```

  - Dictionary is stored in arbitrary order
    - different from what you entered

Dictionary

- **Dictionary is mutable**
  - can change, expand, and shrink them in place without making new dictionaries
  - **del** deletes the entry associated with the key specified as an index

```
>>> D
{'eggs': 3, 'spam': 2, 'ham': 1}
>>> D['ham'] = ['grill', 'bake', 'fry'] # Change entry
>>> D
{'eggs': 3, 'spam': 2, 'ham': ['grill', 'bake', 'fry']}
>>>
>>> del D['eggs']                       # Delete entry
>>> D
{'spam': 2, 'ham': ['grill', 'bake', 'fry']}
>>>
>>> D['brunch'] = 'Bacon'               # Add new entry
>>> D
{'brunch': 'Bacon', 'spam': 2, 'ham': ['grill', 'bake', 'fry']}
```

Dictionary Methods

- **Listing keys or items**

```
>>> D = {'spam': 2, 'ham': 1, 'eggs': 3}
>>> list(D.values())
[3, 2, 1]
>>> list(D.items())
[('eggs', 3), ('spam', 2), ('ham', 1)]
```

- **Accessing non-existent item**

```
>>> D.get('spam')               # A key that is there
2
>>> print(D.get('toast'))   # A key that is missing
None
>>> D.get('toast', 88)
88
```

Dictionary Methods

- **Update**

```
>>> D
{'eggs': 3, 'spam': 2, 'ham': 1}
>>> D2 = {'toast':4, 'muffin':5}
>>> D.update(D2)
>>> D
{'eggs': 3, 'muffin': 5, 'toast': 4, 'spam': 2, 'ham': 1}
```

- **Pop**

```
# pop a dictionary by key
>>> D
{'eggs': 3, 'muffin': 5, 'toast': 4, 'spam': 2, 'ham': 1}
>>> D.pop('muffin')
5
>>> D.pop('toast')          # Delete and return from a key
4
>>> D
{'eggs': 3, 'spam': 2, 'ham': 1}
```

Nested Dictionary

- Dictionary in dictionary

```
>>> rec = {'name': 'Bob',
... 'jobs': ['developer', 'manager'],
... 'web': 'www.bobs.org/Bob',
... 'home': {'state': 'Overworked', 'zip': 12345}}

>>> rec['name']
'Bob'
>>> rec['jobs']
['developer', 'manager']
>>> rec['jobs'][1]
'manager'
>>> rec['home']['zip']
12345
```

Creating Dictionaries

- Several ways to create dictionaries

```
{'name': 'Bob', 'age': 40}   # Traditional literal expression

D = {} # Assign by keys dynamically
D['name'] = 'Bob'
D['age'] = 40

dict(name='Bob', age=40)     # dict keyword argument form

dict([('name', 'Bob'), ('age', 40)])    # dict key/value tuples form
```

Contents

- Function (user defined / built-in)
- Module / Package
- Exceptions

Functions

- **Function types**
  - user defined function
  - python built-in functions (ex. `len()`)
  - functions from the modules / libraries (ex. `math.sqrt()`)

- **User defined function**

```
def func_name (param1, param2, ...):
        ...
        ...


if __name__==‘__main__’:
        ...
        func_name(x, y, ...)
        ...
```

Global vs. Local Variables

- **Global variable**
  - variables available in the program
- **Local variable**
  - variables available only within function regions
  - global variable defined within function : “global” keyword

```
def func_name (param1, param2, ...)
       global a      # global variable within function
       a = 15

       l_num = 10    # local variable
       ...

if __name__==‘__main__’:
       g_num = 20    # global variable
       ...
       func_name(x, y, ...)
       ...
```

Global vs. Local Variables

```
#!/usr/bin/python

# define function
def func():
    global a   # global variable
    a = 10
    b = 15     # local variable

# myprog.py main code
if __name__=="__main__":
    c = 20     # global variable
    func()     # call function

    print(a, b, c)  # where is b ???
```

```
$./myprog.py
Traceback (most recent call last):
      print(a, b, c)  # where is b ???
NameError: name 'b' is not defined
```

Function Parameters

- **Parameters**
  - Fixed sequential parameters
  - Arbitrary argument list

```
def func1 (x,y)
       ...

def func2 (x,y,z)
       ...

def func3 (*args)
       for x in args:
              ...

if __name__==‘__main__’:
       lst = [10,20,30]
       func1(a,b)
       func2(*lst)
       func3(a,b,c,d,..)
```

Function Parameters

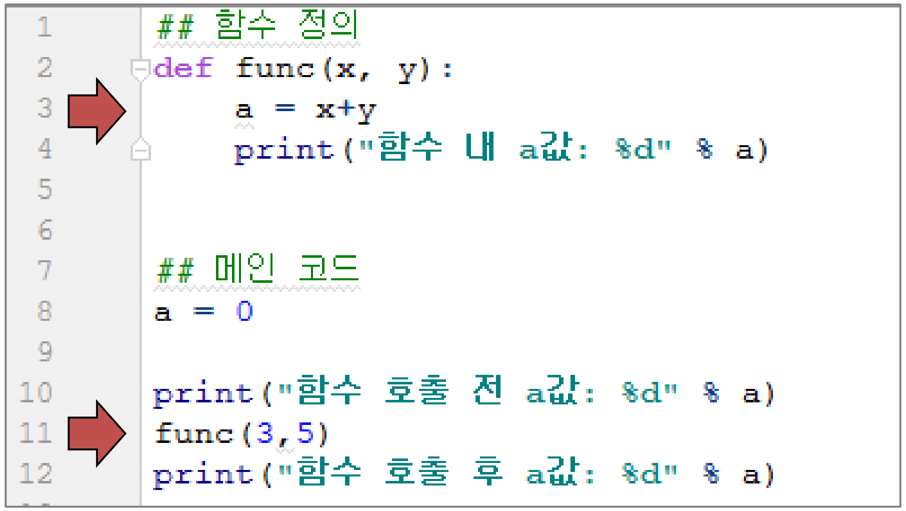
```
#!/usr/bin/python
def func1 (x,y):
    return (x+y)

def func2 (x,y,z):
    return (x+y-z)

def func3 (*args):
    sum = 0
    for num in args:
        sum = sum + num
    return (sum)

if __name__== '__main__':
    a,b,c,d = 1,2,3,4
    lst = [10,20,30]

    print(func1(a,b))
    print(func2(*lst))
    print(func3(a,b,c,d))
```

Call by Value vs. Call by Reference

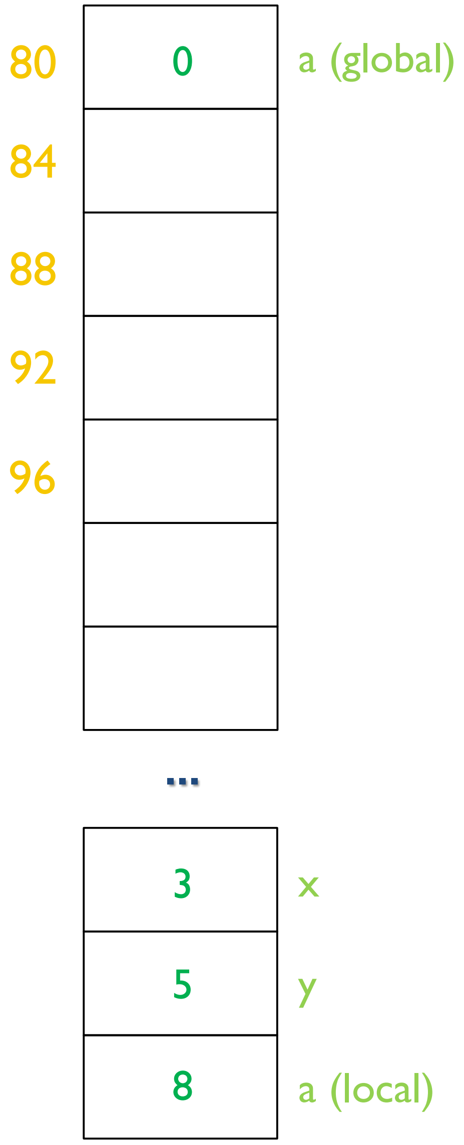
- **Call by Value**
  - the resulting value is bound to the corresponding variable in the function
- **Call by Reference**
  - the function can assign to the variable used as argument.

- **Python arguments are passed using call by value.**
  but! the value is always an <u>object reference</u>, not the value of the object.
  - Actually, call by object reference would be a better description
  - Depending on the data type of the arguments

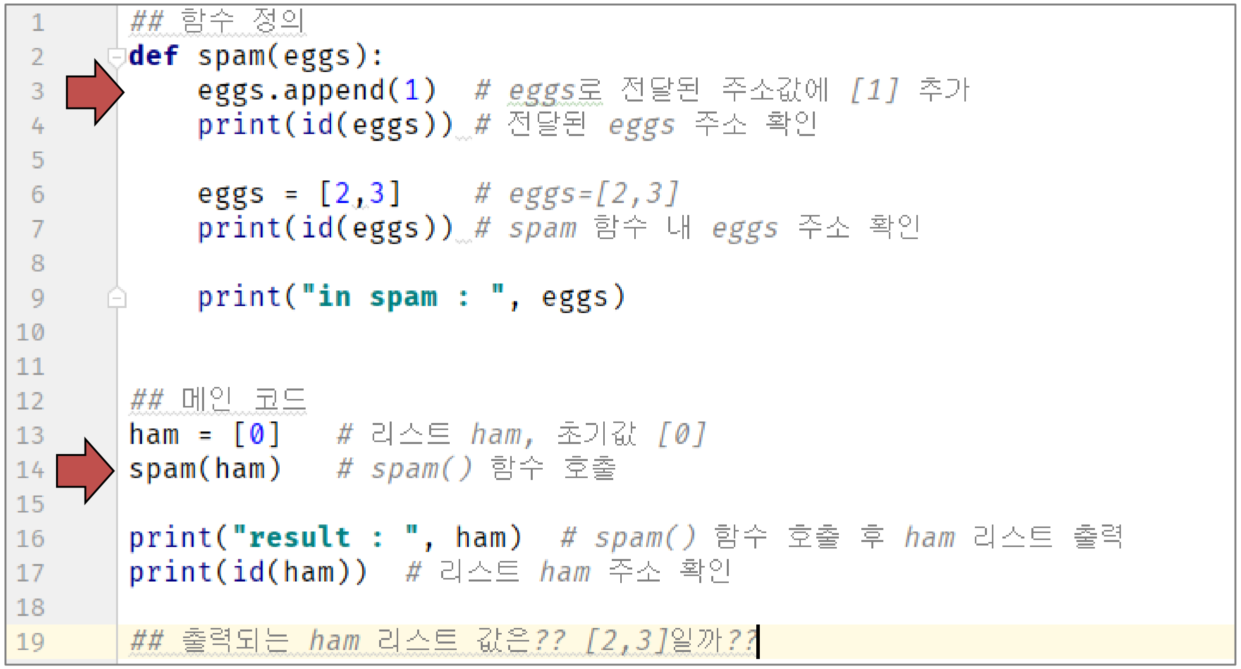
Call by Value



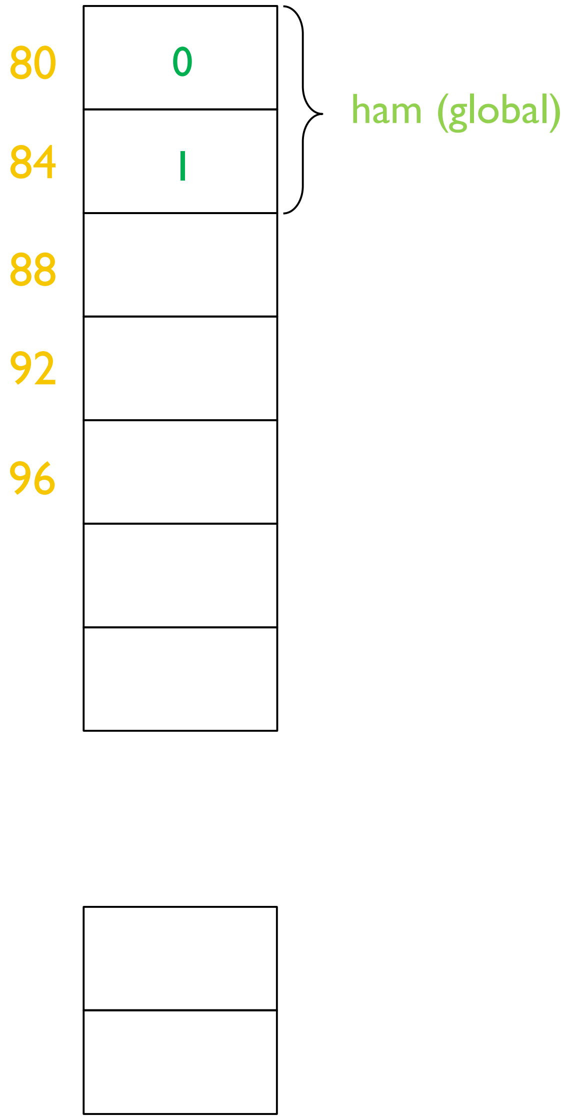
```
## 함수 정의
def func(x, y):
    a = x+y
    print("함수 내 a값: %d" % a)


## 메인 코드
a = 0

print("함수 호출 전 a값: %d" % a)
func(3,5)
print("함수 호출 후 a값: %d" % a)
```



Call by Reference



```
## 함수 정의
def spam(eggs):
    eggs.append(1)  # eggs로 전달된 주소값에 [1] 추가
    print(id(eggs)) # 전달된 eggs 주소 확인

    eggs = [2,3]    # eggs=[2,3]
    print(id(eggs)) # spam 함수 내 eggs 주소 확인

    print("in spam : ", eggs)


## 메인 코드
ham = [0]   # 리스트 ham, 초기값 [0]
spam(ham)   # spam() 함수 호출

print("result : ", ham)  # spam() 함수 호출 후 ham 리스트 출력
print(id(ham))  # 리스트 ham 주소 확인

## 출력되는 ham 리스트 값은?? [2,3]일까??
```



Call by Value

```
#!/usr/bin/python

# define function
def func (x,y):
    a = x+y
    print(a)

# myprog.py main code
if __name__== '__main__':
    a = 0
    print(a)   # pass value
    func(3,5)
    print(a)   # is value changed?
```

```
$./myprog.py
0
8
0
```

Call by Reference

```
#!/usr/bin/python

# define function
def spam(eggs):
    eggs.append(1)
    print("in spam: ", eggs)

# myprog.py main code
if __name__== '__main__':
    ham = [0]
    print(ham)          # pass list
    spam(ham)
    print(ham)          # is list changed?
```

```
$./myprog.py
[0]
in spam:  [0, 1]
[0, 1]
```

Pass Function Object

- **Function is also object.**

```
#!/usr/bin/python
def bye():
    print("bye")

def send(method):
    method()

if __name__== '__main__':
    send(bye)
```

Pass Function Object

- **Function is also object.**

```
#!/usr/bin/python
def plus(a,b):
    return a+b

def minus(a,b):
    return a-b

if __name__== '__main__':
    f_lst = [plus, minus]
    a = f_lst[0](1,2)
    b = f_lst[1](1,2)
    print(a,b)
```

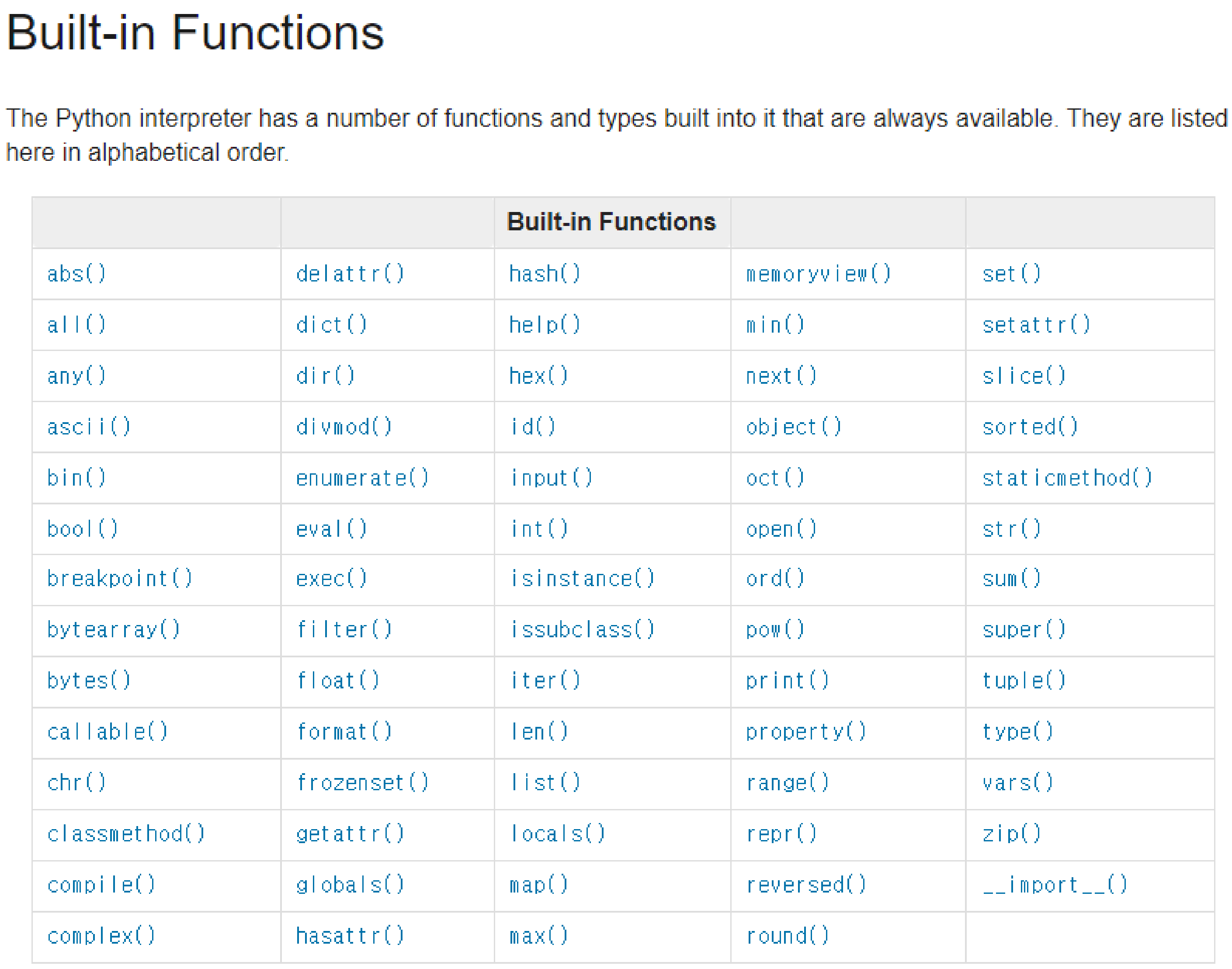
Lambda Expressions

- **Lambda**
  - Small anonymous functions
  - returns the sum of its two arguments: lambda a,b: a+b

```
$ python
>>>
>>> def plus_ten(x):
        return x+10

>>> plus_ten(1)
11
>>>
>>> plus_ten = lambda x: x+10
>>> plus_ten(1)
11
>>>
>>> (lambda x: x+10)(1)
11
>>>
```

Built-in Functions

- https://docs.python.org/3.8/library/functions.html



Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Module

- **A package containing methods that perform specific functions**
  - Standard libraries (built-in modules) / User created module / 3rd party modules
- **Python built-in modules**

```
>>> import sys
>>> print(sys.builtin_module_names)
('_abc', '_ast', '_bisect', '_blake2', '_codecs',
'_collections', '_csv', '_datetime', '_elementtree',
'_functools', '_heapq', '_imp', '_io', '_locale', '_md5',
'_operator', '_pickle', '_posixsubprocess', '_random', '_sha1',
'_sha256', '_sha3', '_sha512', '_signal', '_socket', '_sre',
'_stat', '_statistics', '_string', '_struct', '_symtable',
'_thread', '_tracemalloc', '_warnings', '_weakref', 'array',
'atexit', 'binascii', 'builtins', 'cmath', 'errno',
'faulthandler', 'fcntl', 'gc', 'grp', 'itertools', 'marshal',
'math', 'posix', 'pwd', 'pyexpat', 'select', 'spwd', 'sys',
'syslog', 'time', 'unicodedata', 'xxsubtype', 'zlib')
>>>
```

Module Functions - math

```
from math import *
```

Command name	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1, value2***`)`	larger of two values
`min(`***value1, value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...

Module Functions - random

```
from random import *
```

Command name	Description
`random()`	Return the next random floating point number in the range [0.0, 1.0]
`randint(a,b)`	Return a random integer N such that a <= N <= b

`randrange(stop)`
`randrange(start, stop[, step])`
	Return a randomly selected element from range (start, stop, step)

`seed()`	Initialize the random number generator
`choice([seq])`	Return a random element from the non-empty sequence seq
`shuffle([seq])`	Shuffle the sequence seq in place

Module Functions - time

```
from time import *
```

Command name	Description
`time()`	Return the time in seconds since January 1, 1970, 00:00:00 (UTC) as a floating point number
`localtime()`	Return struct_time of the current time
`localtime(sec)`	Return struct_time of seconds time
`asctime()`	Convert the current time as returned by localtime() 'Sun Jun 20 23:21:05 1993'
`asctime(struct_time)`	Convert a tuple or struct_time representing a time
`sleep(sec)`	Suspend execution of the calling thread for the given number of seconds.

struct_time	Values
`tm_year`	(for example, 1993)
`tm_mon`	range [1, 12]
`tm_mday`	range [1, 31]
`tm_hour`	range [0, 23]
`tm_min`	range [0, 59]
`tm_sec`	range [0, 61]
`tm_wday`	range [0, 6], Monday is 0
`tm_yday`	range [1, 366]
`tm_isdst`	0, 1 or -1 daylight saving time

Module

- **Import user module**

```
import module_name

if __name__==‘__main__’:
       module_name.func()
```

```
import module_name as mn

if __name__==‘__main__’:
       mn.func()
```

```
from module_name import func

if __name__==‘__main__’:
       func()
```

Module Example

- Write user module : calculator.py

```
#!/usr/bin/python
def plus(a,b):
    return a+b

def minus(a,b):
    return a-b

def multiply(a,b):
    return a*b

def divide(a,b):
    return a/b

if __name__== '__main__':                # if no line?
    print("this is my_module...")
```

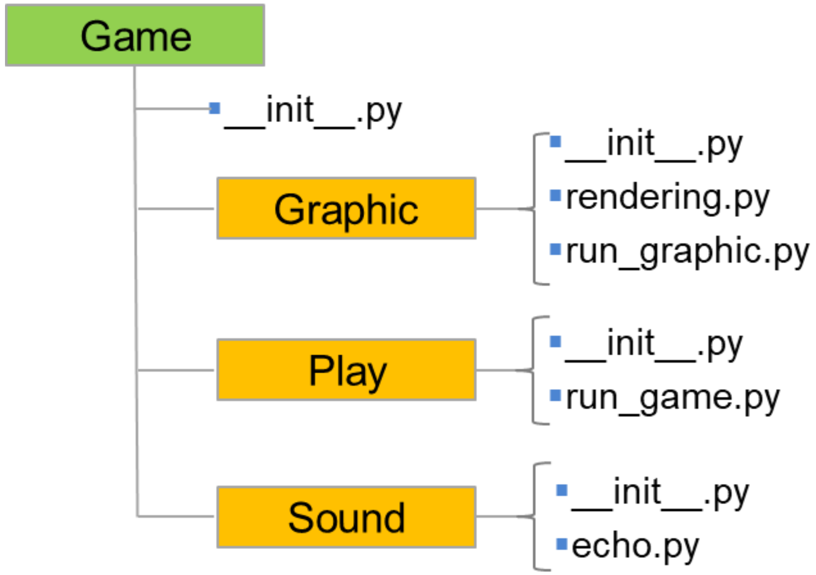
Module Example

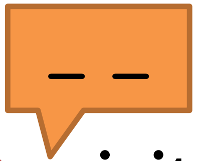
- Import module : myprog.py

```
#!/usr/bin/python
import calculator as cal

if __name__== '__main__':
    num1, num2 = map(int, (input("two numbers:").split()))

    print(cal.plus(num1,num2))
    print(cal.minus(num1,num2))
    print(cal.multiply(num1,num2))
    print(cal.divide(num1,num2))
```

Package

- **directory (package) where modules are collected**





- **__init__.py file is essential for package**
  - After python 3.3 ver, it doesn’t have to be.
  - However, it is recommended to write for version compatibility.

Package Example

- files in *directory* "my_pkg"
  - __init__.py

```
# __init__.py

```

  - calculator.py
  - geometry.py →

```
def triangle_area(base, height):
    return base * height / 2

def rectangle_area(width, height):
    return width * height
```

- myprog.py (main file)

```
#!/usr/bin/python
from my_pkg.calculator import *
from my_pkg.geometry import *

if __name__== '__main__':
    print(my_pkg.plus(10,20))
    print(my_pkg.triangle_area(30,40))
```

Package Example

- files in *directory* "my_pkg"
  - __init__.py

```
# __init__.py

from .calculator import *
from .geometry import *
```

  - calculator.py
  - geometry.py
- myprog.py (main file)

```
#!/usr/bin/python
import my_pkg

if __name__== '__main__':
    print(my_pkg.plus(10,20))
    print(my_pkg.triangle_area(30,40))
```

Exception

- **In Python, all exceptions must be instances of a class that derives from BaseException**

```
try:
    ...
except SomeException:
    tb = sys.exc_info()[2]
    raise OtherException(...).with_traceback(tb)
```

Exception Hierarchy

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      |    +-- RecursionError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
           +-- ResourceWarning
```

Concrete Exceptions

구문에러 (SyntaxError) : 명령의 조건 중 따옴표 오류 등, 구문오류

1) 따옴표나 괄호 닫기 오류

SyntaxError : EOL while scanning string literal
SyntaxError : unexpected EOF while parsing

2) 철자나 따옴표를 빼먹은 경우

SyntaxError : invalid syntax

3) 반복 블록의 들여쓰기 오류

SyntaxError : expected as indented block
SyntaxError : unindent does not match any outer indentation level
SyntaxError : unexpected indent (들여쓰지 말아야 할 곳을 들여쓴 경우)

Concrete Exceptions

이름에러 (NameError) : 명령의 철자 오류

```
TraceBack (most recent call last):
  File "<pyshell#7>", line 1 ,in <module>
    PRINT("Hello")
NameError: name "PRINT" in not defined
```

외부모듈 호출오류 (Import Error) : Import 로 호출 모듈이름 오류

```
TraceBack (most recent call last):
  File "<pyshell#10>", line 11 ,in <module>
    import turtl as t
ImportError: No module named 'turtl'
```

Concrete Exceptions

속성 오류 (AttributeError) : 호출 모듈의 함수, 변수를 잘못 입력

```
TraceBack (most recent call last):
  File "<pyshell#18>", line 21 ,in <module>
     t.forward(50)
AttributeError: 'module' object has no attribute 'forward'
```

타입 에러 (TypeError) : 함수에 전달할 인자가 빠진 경우

```
TypeError : ... missing... required positional argument : ....
```

값 에러 (ValueError) : 정수, 문자 간 값 변환이 불가능 오류

```
ValueError : invalid literal for ... ():
```

Any Questions...

Just Ask!