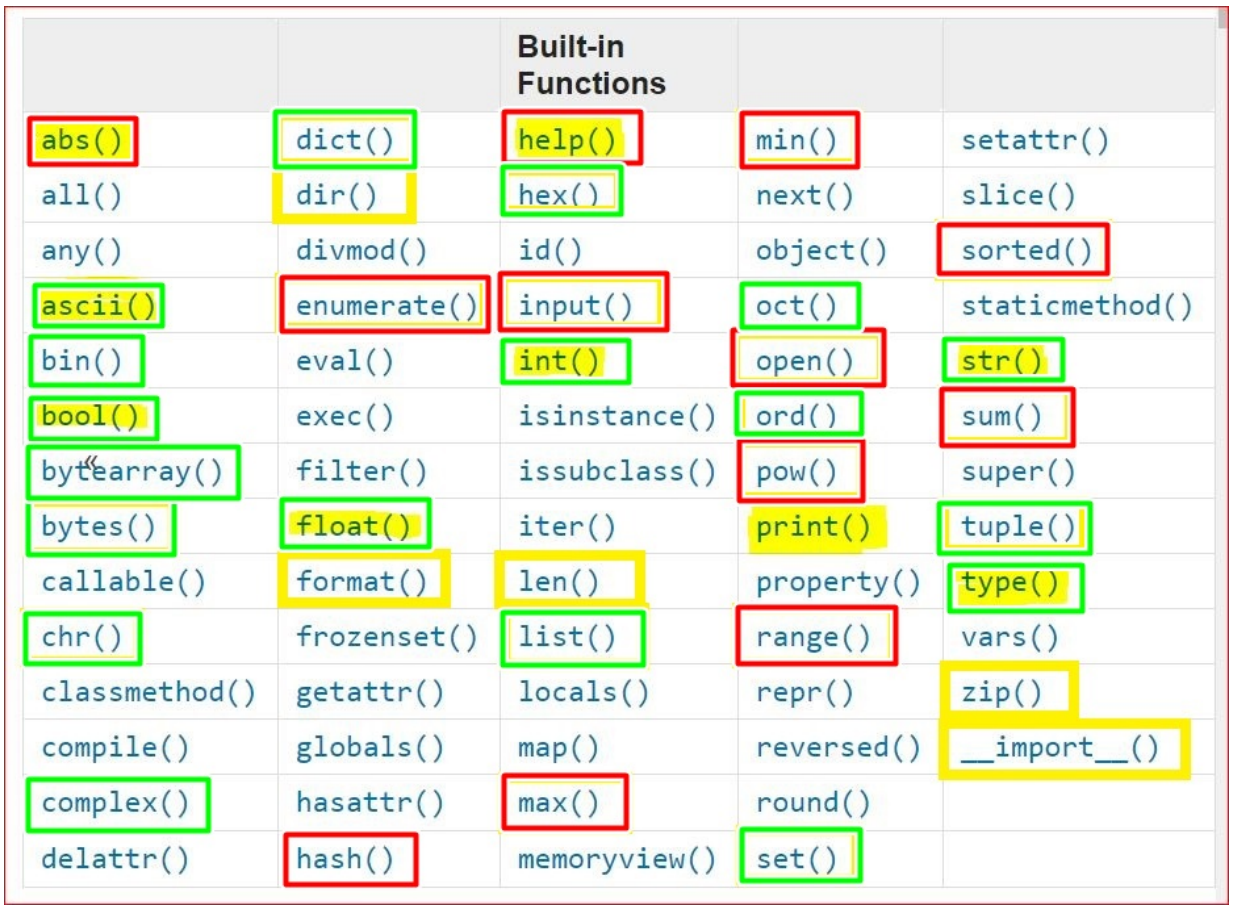
Built-in functions (https://docs.python.org/3/library/functions.html)

In this lesson, we learn many useful built-in functions

https://docs.python.org/3/library/functions.html (https://docs.python.org/3/library/functions.html)



Built-in Functions				
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	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()	
delattr()	hash()	memoryview()	set()	

- Help yourself
  - help()
- I/O - working with files
  - input()
  - open()
- Math - crunching numbers
  - range()
  - abs()
  - min()
  - max()
  - sum()
  - pow()
  - round()
- Useful Others
  - enumerate()

    - sorted()
    - reversed()
    - hash()
- Data Structure and conversion
    - ascii()
    - chr()
    - ord()
    - oct()
    - bin()
    - bool()
    - int()
    - float()
    - complex()
    - bytes()
    - bytearray()
    - str()
    - list()
    - tuple()
    - set()
    - dict()
    - type()

In [1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

Out[1]: run previous cell, wait for 2 seconds

Help

- read online documentation (https://docs.python.org/3/index.html)
- notebook inline help
- ask question at stackoverflow (https://stackoverflow.com/questions/415511/how-to-get-current-time-in-python)

In [2]:
```
# online help
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

ask stackoverflow, for example:

How to get current time in Python? (https://stackoverflow.com/questions/415511/how-to-get-current-time-in-python)

I/O - Input/Output

input() - talk to computer

In [9]:
```
# get inputs from user
your_name = input('What is your name?')
```

```
What is your name?allen
```

In [7]:
```
your_age = input('What is your age?')
```

```
What is your age?13
```

In [8]:
```
your_city = input('Which city are you from?')
```

```
Which city are you from?chapel hill
```

In [10]:
```
print(" name: %s\n age: %s\n city: %s"%(your_name, your_age, your_city))
```

```
 name: allen
 age: 13
 city: chapel hill
```

open, read/write, close - work with files

- read data from file
- write data to file

Syntax of open file:

open(filename, mode)

mode:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

Don't for get to close the file

close()

In [21]:
```
f = open("zen-of-python.txt", "r")
print(type(f))
f.close()
```

```
<class '_io.TextIOWrapper'>
The Zen of Python
```

The read() function

- read() read the file
- read(num) read num of characters
- readline() read a line

In [32]:
```
f = open("zen-of-python.txt", "r")
print(f.readline())
print('============')
print(f.read(3))
print('============')
print(f.read(10))
print('============')
print(f.readline())
print('============')
print(f.read())
f.close()
```

```
The Zen of Python

============
by
============
Tim Peters
============


============

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Simplify the open and close a file

The with ... as ... statement

The with statement will automatically handle file open and close

In [11]:
```
file_zen_python = 'zen-of-python.txt'
with open(file_zen_python, 'r') as f:
    text = f.read()
    print(text)
```

```
The Zen of Python
by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Example: Add line numbers to a text file

In [18]:
```
file_zen_python = 'zen-of-python.txt'
with open(file_zen_python, 'r') as f:
    text = f.read()
    lines = text.split('\n')  # number lines
    n = 0
    for i in lines:
        n = n + 1
        print("[%02d] %s" % (n, i))
```

```
[01] The Zen of Python
[02] by Tim Peters
[03]
[04] Beautiful is better than ugly.
[05] Explicit is better than implicit.
[06] Simple is better than complex.
[07] Complex is better than complicated.
[08] Flat is better than nested.
[09] Sparse is better than dense.
[10] Readability counts.
[11] Special cases aren't special enough to break the rules.
[12] Although practicality beats purity.
[13] Errors should never pass silently.
[14] Unless explicitly silenced.
[15] In the face of ambiguity, refuse the temptation to guess.
[16] There should be one-- and preferably only one --obvious way to do it.
[17] Although that way may not be obvious at first unless you're Dutch.
[18] Now is better than never.
[19] Although never is often better than right now.
[20] If the implementation is hard to explain, it's a bad idea.
[21] If the implementation is easy to explain, it may be a good idea.
[22] Namespaces are one honking great idea -- let's do more of those!
```

Write to a File

Open a file to wirte

open(filename, mode)

mode:

- "a": Append to the end of the file
- "w": Overwrite any existing content in the file

write(data)

In [3]:
```
# write out to a file
filename = 'my-first-file.txt'
f = open(filename, 'w')

f.write("Python is cool.\n") # don't forget the \n as the end of this line
f.write("I like Python.")
f.close()
```

This is what inside the file "my-first-file.txt' :

```
Python is cool
I like Python
```

Delete a File

Use os function remove

In [36]:
```
import os
os.remove("my-first-file.txt")
```

List all of the files in a directory

In [41]:
```
import os
files = os.listdir()
print(type(files))
print(files)
```

```
<class 'list'>
['.ipynb_checkpoints', 'build_in_functions.ipynb', 'Class 8 Python build in
functions.ipynb', 'homework08-solution.ipynb', 'lesson-06-built-in-funcs-fi
le-io', 'lesson-08-functions-modules', 'lesson-08_func_modules.ipynb', 'zen
-of-python.txt']
```

Math - crunching numbers

In [20]:
```
list_1 = list(range(10))
list_1
```

Out[20]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [21]:
```
# math operation on a list of numbers
min(list_1), max(list_1), sum(list_1)
```

Out[21]: (0, 9, 45)

In [22]:
```
float_1 = 2.12345
print(round(float_1))
```

```
2
```

In [23]:
```
# what is 2 to the power of 10?
pow(2,10)
```

Out[23]: 1024

Useful Others

enumerate

In [37]:
```
list_2 = [100, -100, 21, 33, 10, 1000]
# get the index number of a list
enumerate(list_2)
```

Out[37]: <enumerate at 0x1efdcfc8510>

In [26]:
```
list_2 = [100, -100, 21, 33, 10, 1000]
for n,item in enumerate(list_2):
    print("n=%s, item=%s" % (n,item))
```

```
n=0, item=100
n=1, item=-100
n=2, item=21
n=3, item=33
n=4, item=10
n=5, item=1000
```

In [28]:
```
set_1 = {1, 10, 100, 1000}
for n,item in enumerate(set_1):
    print("n=%s, item=%s" % (n,item))
```

```
n=0, item=1000
n=1, item=1
n=2, item=10
n=3, item=100
```

sorted

In [38]:
```
list_2 = [100, -100, 21, 33, 10, 1000]
# sort a list
ordered_list = sorted(list_2)
ordered_list
```

Out[38]: [-100, 10, 21, 33, 100, 1000]

In [30]:
```
rev_order_list = sorted(list_2,reverse=True)
rev_order_list
```

Out[30]: [1000, 100, 33, 21, 10, -100]

In [31]:
```
# did not change the original list
list_2
```

Out[31]: [100, -100, 21, 33, 10, 1000]

In [32]:
```
# sort a list
ordered_list2 = reversed(list_2)
ordered_list2
```

Out[32]: <list_reverseiterator at 0x4c9a470>

In [33]:
```
for i in ordered_list2:
    print(i)
```

```
1000
10
33
21
-100
100
```

Module

Modules are used to group functions, variables, and other things together into larger, more powerful programs.

Module is like Lego

In [ ]:
```

```