# Python

Official Documentation - <https://docs.python.org/3/>

# Data Types and Operator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Structure** | **Ordered** | **Mutable** | **Constructor** | **Example** |
| int | NA | NA | int() | 5 |
| float | NA | NA | float() | 6.5 |
| string | Yes | No | ' ' or " " or str() | "this is a string" |
| bool | NA | NA | NA | True or False |
| list | Yes | Yes | [ ] or list() | [5, 'yes', 5.7] |
| tuple | Yes | No | ( ) or tuple() | (5, 'yes', 5.7) |
| set | No | Yes | { } or set() | {5, 'yes', 5.7} |
| dictionary | No | Keys: No | { } or dict() | {'Jun':75, 'Jul':89} |

# Data type and operator

* Str.spilit() – list of words
* Str.lower()
* Dict.get() – return key value in dict

## dict

* Dict.items()
* Dict.keys()
* Dict.values()

# Function

* Example –

def cylinder\_volume(height, radius=5):

“””doc\_string”””

pi = 3.14159

return height \* pi \* radius \*\* 2

cylinder\_volume(10)

* we can still access the value of the global variable within function. But we cannot modify it
* Lambda expression can be used to create anonymous function -
  + cylinder\_volume = lambda height, radius=5: 3.14159\* height \* radius \*\* 2
* **Iterables** are objects that can return one of their elements at a time, such as a list. An **iterator** is an object that represents a stream of data
* **Generators** are a simple way to create iterators using functions. You can also define iterators using **classes**, (\_\_init\_\_ , \_\_next\_\_ , \_\_iter\_\_[stop Iteration] ) which you can read more about [here](https://docs.python.org/3/tutorial/classes.html#iterators)
* def my\_range(x):

i = 0

while i < x:

yield i “”return values one at a time, and start where it left off each time it’s called””

i += 1

* **Note** - Generators are useful when the fully realized list would not fit in memory, or when the cost to calculate each list element is high and you want to do it as late as possible. But they can only be iterated over once.
* **Generator expressions and list comprehensions**

line\_list = [' line 1\n', 'line 2 \n', ...]

####### Generator expression -- returns iterator (surrounded by parentheses (“()”))

stripped\_iter = (line.strip() for line in line\_list)

####### List comprehension -- returns list (surrounded by square brackets (“[]”))

stripped\_list = [line.strip() for line in line\_list]