



## 1

## 2



## Basic characteristics

- Python is a free and open source language, first written in C.
- Python is an interpreted language.





- Python is a free and open source language, **first written in C.**
- Python is an interpreted language.
- Python is **not** a free-form language.
- Python is a strongly typed language.
- Python is an object-oriented language **but it also supports procedural oriented programming.**









1. *Journal of the American Medical Association*, 1997; 277: 1039-1043.



◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡



# Knowing Python in a better way

## Looking into the help manual:

```
help(<built-in object>)
help(<built-in function>)
```

## Looking into the source code:

```
import inspect
inspect.getfullargspec(<built-in function>)
inspect.getsource(<live object>)
```

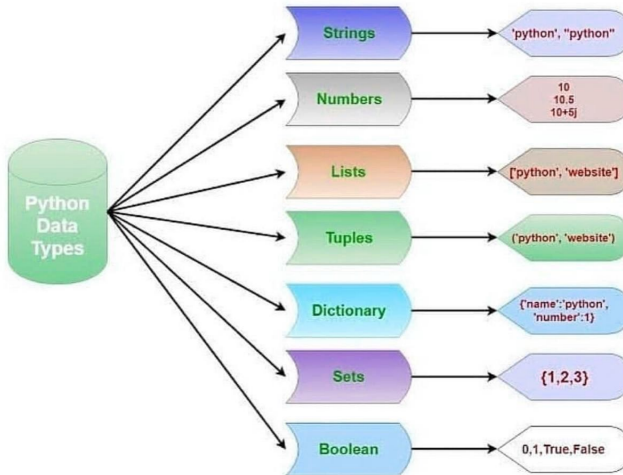
**Note:** The official Python source releases are available at: <https://www.python.org/downloads/source>.

# Knowing Python in a better way

The full standard library documentation of Python 3 can be found at: <https://docs.python.org/3/contents.html>



# The data types in Python



◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

## Introduction to Computing

### Substring of a string:

### Output:

## Splitting a string:

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

# Strings

### Substring of a string:

```
str = "malfunctioning" # Indexing starts with 0
print(str[3:6]) # Elements indexed 3 through 5
```

### Output:

fun

## Splitting a string:

```
str = "I don't love Python, the snake."
print(str.split(" "), str.split(", "))
```

### Output:

```
['I', "don't", 'love', 'Python,', 'the', 'snake.']  
["I don't love Python", 'the snake.']
```

1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 26

$$\| \mathbf{p} \| = \| \mathbf{p} \mathbf{p}^T \mathbf{p} \|$$

# Strings

What is the connection between the outputs of the following two cases?

```
str = "BStat_I"
print(hex(id(str)))    print(hex(id("BStat" + "_" + "I")))
```

They are the same!!!

**Note:** `id()` is used to return the identity of an object in Python.

## Negative indexing:

```
str = 'please step on no pets'
strrev = str[::-1]
print(strrev)
```



◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

# Strings

Python 3 standard library documentation on the common string operations can be found at:

<https://docs.python.org/3/library/string.html>

The source code for the string module is available here: <https://github.com/python/cpython/tree/3.11/Lib/string.py>

• (1)

int(1)	1
1.	+3.45e67
float(1)	+3.45e-67
.1	-3.45e-67
1.2345	3.45e67
-0.6789	.00345e-32
+.4560	1e-15
-.1234	1e+15
complex(1,-1)	(1-1j)

• (1)

int(1)	1
1.	+3.45e67
float(1)	+3.45e-67
.1	-3.45e-67
1.2345	3.45e67
-0.6789	.00345e-32
+.4560	1e-15
-.1234	1e+15
complex(1,-1)	(1-1j)

- Do not use commas as thousand-separators.
- At times behavior may be counter-intuitive.

# Numbers

What will be the output of the following?

a = 10

b = 10

c = 3+7

d = 11

e = 12

```
print(hex(id(a)), hex(id(b)), hex(id(c)))
```

```
print(hex(id(d)), hex(id(e)))
```

**Output:**

0x7fe59418c210 0x7fe59418c210 0x7fe59418c210

0x7fe59418c230 0x7fe59418c250

# Numbers

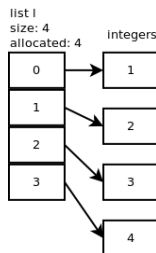
What will be the output of the following?

```
a = complex(1,-1)
b = 1-1j
print(a,b)
print(hex(id(a)), hex(id(b)))
c = b * complex(1,1)
d = 2
print(c,d)
print(hex(id(c)), hex(id(d)))
```

**Output:**

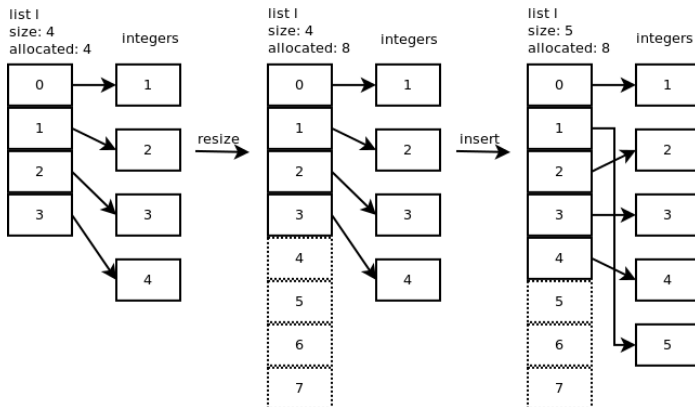
```
(1-1j) (1-1j)
0x7f7f6445a770 0x7f7f6445a6b0
(2+0j) 2
0x7f7f6445a6f0 0x7f7f645bc110
```

# Lists



- *Length* of the list =  $n$ .
- $n$  *linked* memory locations that gets dynamically reallocated.
- The elements are heterogeneous.
- *Elements* can be mapped to each of the  $n$  memory locations.
- Elements are indexed 0 through  $n - 1$  ( $\equiv -n$  through  $-1$ ).

# Lists – Adding elements



```
ls = []
```

```
ls.append(<data>) # Inserts at the end
```

```
ls.insert(<index>, <data>) # Inserts at the <index>
```



# Lists – Adding elements

What will be the output of the following program?

```
ls = [1, 2, 3, 4, 5]
ls.insert(3, 20) # Inserts 20 at index 3
print(ls)
ls.insert(20, 3) # Inserts 3 at index 6 (not at index 20)
print(ls)
```

# Lists – Adding elements

What will be the output of the following program?

```
ls = [1, 2, 3, 4, 5]
ls.insert(3, 20) # Inserts 20 at index 3
print(ls)
ls.insert(20, 3) # Inserts 3 at index 6 (not at index 20)
print(ls)
```

**Output:**

```
[1, 2, 3, 20, 4, 5]
[1, 2, 3, 20, 4, 5, 3]
```

**Note:** If the list insertion index is out of range then the maximum possible range is taken.

# Lists – Adding elements

What will be the output of the following program?

```
ls = [1, 2, 3, 4, 5]
ls[3] = 10 # Assigns 10 at index 3
print(ls)
ls[10] = 3 # Assigns 3 at index 10
print(ls)
```

# Lists – Adding elements

What will be the output of the following program?

```
ls = [1, 2, 3, 4, 5]
ls[3] = 10 # Assigns 10 at index 3
print(ls)
ls[10] = 3 # Assigns 3 at index 10
print(ls)
```

**Output:**

```
[1, 2, 3, 10, 5]
```

**Error**

**Note:** List assignment index cannot be out of range.

# Lists – Adding multiple elements

```
ls1 = [1, 2, 3, 4]
ls2 = [8, 9, 10, 11]
ls1.extend(ls2) # Extends ls1 by appending ls2
print(ls1)
ls1[4:4] = [5, 6, 7] # Inserts elements from a list
print(ls1)
ls1[4] = [4.5, 5, 5.5] # Inserts a list
print(ls1)
```

## Lists – Adding multiple elements

```
ls1 = [1, 2, 3, 4]
ls2 = [8, 9, 10, 11]
ls1.extend(ls2) # Extends ls1 by appending ls2
print(ls1)
ls1[4:4] = [5, 6, 7] # Inserts elements from a list
print(ls1)
ls1[4] = [4.5, 5, 5.5] # Inserts a list
print(ls1)
```

### Output:

```
[1, 2, 3, 4, 8, 9, 10, 11]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[1, 2, 3, 4, [4.5, 5, 5.5], 6, 7, 8, 9, 10, 11]
```

# Lists – Adding multiple elements

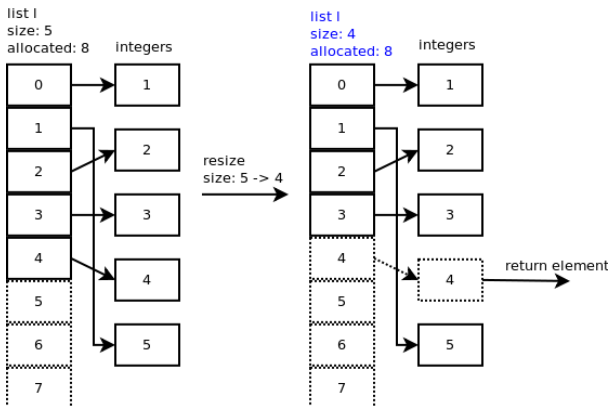
```
ls1 = [1, 2, 3, 4]
ls2 = [8, 9, 10, 11]
ls1.extend(ls2) # Extends ls1 by appending ls2
print(ls1)
ls1[4:4] = [5, 6, 7] # Inserts elements from a list
print(ls1)
ls1[4] = [4.5, 5, 5.5] # Inserts a list
print(ls1)
```

## Output:

```
[1, 2, 3, 4, 8, 9, 10, 11]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[1, 2, 3, 4, [4.5, 5, 5.5], 6, 7, 8, 9, 10, 11]
```

**Note:** You may use an iterative control flow.

# Lists – Removing elements



```
ls = []
```

```
<data> = ls.pop() # Deletes from the end
```

```
<data> = ls.pop(<index>) # Deletes from the <index>
```

```
<data> = ls.remove(<index>) # Deletes from the <index>
```



# Lists – Removing multiple elements

You may use an iterative control flow.

# Lists

## Creating list of lists:

```
ls = [[]] * 5  
print(ls)
```

# Lists

## Creating list of lists:

```
ls = [[]] * 5  
print(ls)
```

## Output:

```
[[], [], [], [], []]
```

# Lists of sublists

What will be the output of the following program?

```
ls1 = [[] for i in range(3)]
ls1[0].append(10)
ls1[1].append(20)
ls1.append(30)
print(ls1)
ls2 = [[]]*3
ls2[0].append(10)
ls2[1].append(20)
ls2.append(30)
print(ls2)
```

# Lists of sublists

What will be the output of the following program?

```
ls1 = [[] for i in range(3)]  
ls1[0].append(10)  
ls1[1].append(20)  
ls1.append(30)  
print(ls1)  
ls2 = [[]]*3  
ls2[0].append(10)  
ls2[1].append(20)  
ls2.append(30)  
print(ls2)
```

```
[[10], [20], [], 30]  
[[10, 20], [10, 20], [10, 20], 30]
```

# Lists

Python Standard Library documentation on the extended list operations can be found at: <https://docs.python.org/3/tutorial/datastructures.html>

# Converting Strings to Lists

Strings are just like Lists. One can convert a String into the List by passing the String as an argument to the List as follows.

```
str = "Python 3"  
list(str)
```

**Output:** ['P', 'y', 't', 'h', 'o', 'n', ' ', '3']

# Tuples

Tuples are like lists but immutable in nature, i.e. the elements in the tuple cannot be added or removed once created.

```
tp = ('Language', 'Python')  
print(tp, tp[1])
```

**Output:** ('Language', 'Python') Python



# Tuples

Tuples are like lists but immutable in nature, i.e. the elements in the tuple cannot be added or removed once created.

```
tp = ('Language', 'Python')  
print(tp, tp[1])
```

**Output:** ('Language', 'Python') Python

## Converting lists to tuples:

```
ls = [1, 2, 4, 5, 6]  
Tuple = tuple(ls)
```

# Tuples

Python Standard Library documentation on the extended tuple operations can be found at: <https://docs.python.org/3/tutorial/datastructures.html>

# Dictionary

## Creating a Dictionary:

```
dc = {'Course': 'Business Analytics', 1: [28, 29]}  
print("The created dictionary: ", dc)
```

## **Output:**

The created dictionary: 'Course': 'Business Analytics', 1: [28, 29]

# Dictionary

## Accessing an element from the dictionary:

```
print(dc['Course'], dc[1]) # Accessed by the key  
print(dc.get('Course'), dc.get(1))
```

## Output:

```
Business Analytics [28, 29]
```

```
Business Analytics [28, 29]
```

# Dictionary

## Accessing an element from the dictionary:

```
print(dc['Course'], dc[1]) # Accessed by the key  
print(dc.get('Course'), dc.get(1))
```

### Output:

```
Business Analytics [28, 29]  
Business Analytics [28, 29]
```

## Deleting an element from the dictionary:

```
dc.pop('Course')  
print("The current dictionary: ", dc)
```

### Output:

```
The current dictionary: 1: [28, 29]
```

# Dictionary

Python Standard Library documentation on the extended dictionary operations can be found at: <https://docs.python.org/3/tutorial/datastructures.html>

# Sets

Sets are (ordered or unordered) collection of data items that is mutable and does not allow any duplicate element.

```
st = {'day', 1, 2, 'for', 'Python', 'Python'}  
print(st)  
st.add('language')  
print(st)  
st.remove('for')  
print(st)
```

## Output:

```
{'Python', 1, 2, 'for', 'day'}  
{'Python', 1, 2, 'for', 'day', 'language'}  
{'Python', 1, 2, 'day', 'language'}
```

**Note:** Sets are not subscriptable.

# Sets

Python Standard Library documentation on the extended set operations can be found at: <https://docs.python.org/3/tutorial/datastructures.html>



# Boolean values

Any non-zero value is treated as True and zero is treated as False.

## Examples:

0	False	0e10	False
1	True	'A'	True
6 - 2 * 3	False	"A"	True
(6 - 2) * 3	True	'\0'	True
0.0075	True	(0, 0)	True
"	False	""	False

# Boolean values

Any non-zero value is treated as True and zero is treated as False.

## Examples:

0	False	0e10	False
1	True	'A'	True
6 - 2 * 3	False	"A"	True
(6 - 2) * 3	True	'\0'	True
0.0075	True	(0, 0)	True
"	False	""	False

**Note:** The expressions like "x = 0" or "x = 1" will exhibit error.

# Mutable and immutable objects

Everything in Python is an object and it is either mutable or immutable.

# Mutable and immutable objects

Everything in Python is an object and it is either mutable or immutable.

A mutable object can be changed to a different type after it is created, but an immutable object cannot be changed.

# Mutable and immutable objects

Everything in Python is an object and it is either mutable or immutable.

A mutable object can be changed to a different type after it is created, but an immutable object cannot be changed.

- Objects of built-in types like int, float, bool, str, tuple, unicode are immutable.
- Objects of built-in types like list, set, dict are mutable.

# Verifying the data type

```
type(3)
```

**Output:** int

# Verifying the data type

```
type(3)
```

**Output:** int

```
type(3.14)
```

**Output:** float

**Output:** int

**Output:** float

**Output:** str



# Verifying the data type

```
type(3)
```

**Output:** int

```
type(3.14)
```

**Output:** float

```
type('pi')
```

**Output:** str

```
type([3.14, 3.142, 3.1428])
```

**Output:** list

# Verifying the data type

```
type(3)
```

**Output:** int

```
type(3.14)
```

**Output:** float

```
type('pi')
```

**Output:** str

```
type([3.14, 3.142, 3.1428])
```

**Output:** list

```
type(True)
```

**Output:** bool