**LECTURE TWO**

**Chapter 1:Strings**

* In python Strings are created by either using single quotes or double quotes
* To display them on the screen we shall use print() function.

Example

print("Hello")  
print('Hello')

Assigning variables to string

* In order to assign a string to a variable in python we use a variable name followed by and equal sign lastly we incorporate a string literal

Example

a = "Hello"  
print(a)

Assign multiline string

* You can assign multiline string to a variable with ““”string “””

Example

a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)

Accessing values of string

* Strings in python are array of bits representing Unicode character
* However python does not have character datatype thus this are treated as string with length one.
* To access a value in a string use square bracket for slicing along with index or indices to obtain your substrings

Example

a = "Hello, World!"  
print(a[1])

Example2

b = "Hello, World!"  
print(b[2:5])

Example3

b = "Hello, World!"  
print(b[-5:-2])

Escape Characters

* Following table is a list of escape or non-printable characters that can be represented with backslash notation.
* An escape character gets interpreted; in a single quoted as well as double quoted strings.

|  |  |  |
| --- | --- | --- |
| **Backslash 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 |
| \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

* Assume string variable **a** holds 'Hello' and variable **b** holds 'Python', then −

|  |  |  |
| --- | --- | --- |
| **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 |
| 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 |
| r/R | Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark. | print r'\n' prints \n and print R'\n'prints \n |
| % | Format - Performs String formatting | See at next section |

## String Formatting Operator

## This is one of the coolest feature in python where we use the operator % to format various strings

## This operator is unique to strings and makes up for the pack of having functions from C's printf() family

Example

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

* We can also achieve formatting by the use of format() method.

Example

age = 36  
txt = "My name is John, and I am {}"  
print(txt.format(age))

Example2

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want {} pieces of item {} for {} dollars."  
print(myorder.format(quantity, itemno, price))

Example3

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."  
print(myorder.format(quantity, itemno, price))

* Here is the list of complete set of symbols which can be used along with % −

|  |  |
| --- | --- |
| **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 |

* Other supported symbols and functionality are listed in the following table −

|  |  |
| --- | --- |
| **Symbol** | **Functionality** |
| \* | 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. |
| 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.) |

## String Methods

|  |  |
| --- | --- |
| **Sr.No.** | **Methods with Description** |
| 1 | [capitalize()](https://www.tutorialspoint.com/python/string_capitalize.htm)  Capitalizes first letter of string |
| 2 | [center(width, fillchar)](https://www.tutorialspoint.com/python/string_center.htm)  Returns a space-padded string with the original string centered to a total of width columns. |
| 3 | [count(str, beg= 0,end=len(string))](https://www.tutorialspoint.com/python/string_count.htm)  Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given. |
| 4 | [decode(encoding='UTF-8',errors='strict')](https://www.tutorialspoint.com/python/string_decode.htm)  Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding. |
| 5 | [encode(encoding='UTF-8',errors='strict')](https://www.tutorialspoint.com/python/string_encode.htm)  Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'. |
| 6 | [endswith(suffix, beg=0, end=len(string))](https://www.tutorialspoint.com/python/string_endswith.htm)  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. |
| 7 | [expandtabs(tabsize=8)](https://www.tutorialspoint.com/python/string_expandtabs.htm)  Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided. |
| 8 | [find(str, beg=0 end=len(string))](https://www.tutorialspoint.com/python/string_find.htm)  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. |
| 9 | [index(str, beg=0, end=len(string))](https://www.tutorialspoint.com/python/string_index.htm)  Same as find(), but raises an exception if str not found. |
| 10 | [isalnum()](https://www.tutorialspoint.com/python/string_isalnum.htm)  Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise. |
| 11 | [isalpha()](https://www.tutorialspoint.com/python/string_isalpha.htm)  Returns true if string has at least 1 character and all characters are alphabetic and false otherwise. |
| 12 | [isdigit()](https://www.tutorialspoint.com/python/string_isdigit.htm)  Returns true if string contains only digits and false otherwise. |
| 13 | [islower()](https://www.tutorialspoint.com/python/string_islower.htm)  Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise. |
| 14 | [isnumeric()](https://www.tutorialspoint.com/python/string_isnumeric.htm)  Returns true if a unicode string contains only numeric characters and false otherwise. |
| 15 | [isspace()](https://www.tutorialspoint.com/python/string_isspace.htm)  Returns true if string contains only whitespace characters and false otherwise. |
| 16 | [istitle()](https://www.tutorialspoint.com/python/string_istitle.htm)  Returns true if string is properly "titlecased" and false otherwise. |
| 17 | [isupper()](https://www.tutorialspoint.com/python/string_isupper.htm)  Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise. |
| 18 | [join(seq)](https://www.tutorialspoint.com/python/string_join.htm)  Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string. |
| 19 | [len(string)](https://www.tutorialspoint.com/python/string_len.htm)  Returns the length of the string |
| 20 | [ljust(width[, fillchar])](https://www.tutorialspoint.com/python/string_ljust.htm)  Returns a space-padded string with the original string left-justified to a total of width columns. |
| 21 | [lower()](https://www.tutorialspoint.com/python/string_lower.htm)  Converts all uppercase letters in string to lowercase. |
| 22 | [lstrip()](https://www.tutorialspoint.com/python/string_lstrip.htm)  Removes all leading whitespace in string. |
| 23 | [maketrans()](https://www.tutorialspoint.com/python/string_maketrans.htm)  Returns a translation table to be used in translate function. |
| 24 | [max(str)](https://www.tutorialspoint.com/python/string_max.htm)  Returns the max alphabetical character from the string str. |
| 25 | [min(str)](https://www.tutorialspoint.com/python/string_min.htm)  Returns the min alphabetical character from the string str. |
| 26 | [replace(old, new [, max])](https://www.tutorialspoint.com/python/string_replace.htm)  Replaces all occurrences of old in string with new or at most max occurrences if max given. |
| 27 | [rfind(str, beg=0,end=len(string))](https://www.tutorialspoint.com/python/string_rfind.htm)  Same as find(), but search backwards in string. |
| 28 | [rindex( str, beg=0, end=len(string))](https://www.tutorialspoint.com/python/string_rindex.htm)  Same as index(), but search backwards in string. |
| 29 | [rjust(width,[, fillchar])](https://www.tutorialspoint.com/python/string_rjust.htm)  Returns a space-padded string with the original string right-justified to a total of width columns. |
| 30 | [rstrip()](https://www.tutorialspoint.com/python/string_rstrip.htm)  Removes all trailing whitespace of string. |
| 31 | [split(str="", num=string.count(str))](https://www.tutorialspoint.com/python/string_split.htm)  Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given. |
| 32 | [splitlines( num=string.count('\n'))](https://www.tutorialspoint.com/python/string_splitlines.htm)  Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed. |
| 33 | [startswith(str, beg=0,end=len(string))](https://www.tutorialspoint.com/python/string_startswith.htm)  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. |
| 34 | [strip([chars])](https://www.tutorialspoint.com/python/string_strip.htm)  Performs both lstrip() and rstrip() on string. |
| 35 | [swapcase()](https://www.tutorialspoint.com/python/string_swapcase.htm)  Inverts case for all letters in string. |
| 36 | [title()](https://www.tutorialspoint.com/python/string_title.htm)  Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase. |
| 37 | [translate(table, deletechars="")](https://www.tutorialspoint.com/python/string_translate.htm)  Translates string according to translation table str(256 chars), removing those in the del string. |
| 38 | [upper()](https://www.tutorialspoint.com/python/string_upper.htm)  Converts lowercase letters in string to uppercase. |
| 39 | [zfill (width)](https://www.tutorialspoint.com/python/string_zfill.htm)  Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero). |
| 40 | [isdecimal()](https://www.tutorialspoint.com/python/string_isdecimal.htm)  Returns true if a unicode string contains only decimal characters and false otherwise. |

**CHAPTER2 : BasicOperators**

Types of Operator

Python language supports the following types of operators.

* Arithmetic Operators
* Comparison (Relational) Operators
* Assignment Operators
* Logical Operators
* Bitwise Operators
* Membership Operators
* Identity Operators

Let us have a look on all operators one by one.

Python Arithmetic Operators

* Assume variable a holds 10 and variable b holds 20, then −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + Addition | Adds values on either side of the operator. | a + b = 30 |
| - Subtraction | Subtracts right hand operand from left hand operand. | a – b = -10 |
| \* Multiplication | Multiplies values on either side of the operator | a \* b = 200 |
| / Division | Divides left hand operand by right hand operand | b / a = 2 |
| % Modulus | Divides left hand operand by right hand operand and returns remainder | b % a = 0 |
| \*\* Exponent | Performs exponential (power) calculation on operators | a\*\*b =10 to the power 20 |
| // | Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) − | 9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0 |

Python Comparison Operators

* These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.
* Assume variable a holds 10 and variable b holds 20, then −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | If the values of two operands are equal, then the condition becomes true. | (a == b) is not true. |
| != | If values of two operands are not equal, then condition becomes true. | (a != b) is true. |
| <> | If values of two operands are not equal, then condition becomes true. | (a <> b) is true. This is similar to != operator. |
| > | If the value of left operand is greater than the value of right operand, then condition becomes true. | (a > b) is not true. |
| < | If the value of left operand is less than the value of right operand, then condition becomes true. | (a < b) is true. |
| >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. | (a >= b) is not true. |
| <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. | (a <= b) is true. |

Python Assignment Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Assigns values from right side operands to left side operand | c = a + b assigns value of a + b into c |
| += Add AND | It adds right operand to the left operand and assign the result to left operand | c += a is equivalent to c = c + a |
| -= Subtract AND | It subtracts right operand from the left operand and assign the result to left operand | c -= a is equivalent to c = c - a |
| \*= Multiply AND | It multiplies right operand with the left operand and assign the result to left operand | c \*= a is equivalent to c = c \* a |
| /= Divide AND | It divides left operand with the right operand and assign the result to left operand | c /= a is equivalent to c = c / a |
| %= Modulus AND | It takes modulus using two operands and assign the result to left operand | c %= a is equivalent to c = c % a |
| \*\*= Exponent AND | Performs exponential (power) calculation on operators and assign value to the left operand | c \*\*= a is equivalent to c = c \*\* a |
| //= Floor Division | It performs floor division on operators and assign value to the left operand | c //= a is equivalent to c = c // a |

Python Bitwise Operators

* Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively. Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands −

a = 0011 1100

b = 0000 1101

-----------------

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a  = 1100 0011

There are following Bitwise operators supported by Python language

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & Binary AND | Operator copies a bit to the result if it exists in both operands | (a & b) (means 0000 1100) |
| | Binary OR | It copies a bit if it exists in either operand. | (a | b) = 61 (means 0011 1101) |
| ^ Binary XOR | It copies the bit if it is set in one operand but not both. | (a ^ b) = 49 (means 0011 0001) |
| ~ Binary Ones Complement | It is unary and has the effect of 'flipping' bits. | (~a ) = -61 (means 1100 0011 in 2's complement form due to a signed binary number. |
| << Binary Left Shift | The left operands value is moved left by the number of bits specified by the right operand. | a << 2 = 240 (means 1111 0000) |
| >> Binary Right Shift | The left operands value is moved right by the number of bits specified by the right operand. | a >> 2 = 15 (means 0000 1111) |

Python Logical Operators

* There are following logical operators supported by Python language.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| and Logical AND | If both the operands are true then condition becomes true. | (a and b) is true. |
| or Logical OR | If any of the two operands are non-zero then condition becomes true. | (a or b) is true. |
| not Logical NOT | Used to reverse the logical state of its operand. | Not(a and b) is false. |

Python Membership Operators

Python’s membership operators test for membership in a sequence, such as strings, lists, or tuples.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| in | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | x in y, here in results in a 1 if x is a member of sequence y. |
| not in | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. | x not in y, here not in results in a 1 if x is not a member of sequence y. |

Python Identity Operators

* Identity operators compare the memory locations of two objects.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| is | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. | x is y, here **is** results in 1 if id(x) equals id(y). |
| is not | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. | x is not y, here **is not** results in 1 if id(x) is not equal to id(y). |

Python Operators Precedence

* The following table lists all operators from highest precedence to lowest.

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | **\*\***  Exponentiation (raise to the power) |
| 2 | **~ + -**  Complement, unary plus and minus (method names for the last two are +@ and -@) |
| 3 | **\* / % //**  Multiply, divide, modulo and floor division |
| 4 | **+ -**  Addition and subtraction |
| 5 | **>> <<**  Right and left bitwise shift |
| 6 | **&**  Bitwise 'AND' |
| 7 | **^ |**  Bitwise exclusive `OR' and regular `OR' |
| 8 | **<= < > >=**  Comparison operators |
| 9 | **<> == !=**  Equality operators |
| 10 | **= %= /= //= -= += \*= \*\*=**  Assignment operators |
| 11 | **is is not**  Identity operators |
| 12 | **in not in**  Membership operators |
| 13 | **not or and**  Logical operators |

**CHAPTER3:LIST**

* List is a collection which is ordered and changeably.
* Allows duplicate members
* To create a list we usually assign a list to a variable using [] brackets.

Example

thislist = ["apple", "banana", "cherry"]  
print(thislist)

Accessiong items in a list

* List indexing starts from element 0 which represents the first element going to 1 which represents the next element etc.
* Example1

thislist = ["apple", "banana", "cherry"]  
 print(thislist[1])

Example2

list1 = ['physics', 'chemistry', 1997, 2000]

list2 = [1, 2, 3, 4, 5, 6, 7 ]

print ("list1[0]: ", list1[0])

print ("list2[1:5]: ", list2[1:5])

Example3

thislist = ["apple", "banana", "cherry"]  
print(thislist[-1])

Updating elements in a list

Example

thislist = ["apple", "banana", "cherry"]  
thislist[1] = "blackcurrant"  
print(thislist)

Loop through a list

thislist = ["apple", "banana", "cherry"]  
for x in thislist:  
  print(x)

Check if Item Exists

thislist = ["apple", "banana", "cherry"]  
if "apple" in thislist:  
  print("Yes, 'apple' is in the fruits list")

Join Two Lists

list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
list3 = list1 + list2  
print(list3)

Basic List Operations

Lists respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

|  |  |  |
| --- | --- | --- |
| **Python Expression** | **Results** | **Description** |
| len([1, 2, 3]) | 3 | Length |
| [1, 2, 3] + [4, 5, 6] | [1, 2, 3, 4, 5, 6] | Concatenation |
| ['Hi!'] \* 4 | ['Hi!', 'Hi!', 'Hi!', 'Hi!'] | Repetition |
| 3 in [1, 2, 3] | True | Membership |
| for x in [1, 2, 3]: print x, | 1 2 3 | Iteration |

Built-in List Functions & Methods

|  |  |
| --- | --- |
| **Sr.No.** | **Function with Description** |
| 1 | [cmp(list1, list2)](https://www.tutorialspoint.com/python/list_cmp.htm)  Compares elements of both lists. |
| 2 | [len(list)](https://www.tutorialspoint.com/python/list_len.htm)  Gives the total length of the list. |
| 3 | [max(list)](https://www.tutorialspoint.com/python/list_max.htm)  Returns item from the list with max value. |
| 4 | [min(list)](https://www.tutorialspoint.com/python/list_min.htm)  Returns item from the list with min value. |
| 5 | [list(seq)](https://www.tutorialspoint.com/python/list_list.htm)  Converts a tuple into list. |

Python includes following list methods

|  |  |
| --- | --- |
| **Sr.No.** | **Methods with Description** |
| 1 | [list.append(obj)](https://www.tutorialspoint.com/python/list_append.htm)  Appends object obj to list |
| 2 | [list.count(obj)](https://www.tutorialspoint.com/python/list_count.htm)  Returns count of how many times obj occurs in list |
| 3 | [list.extend(seq)](https://www.tutorialspoint.com/python/list_extend.htm)  Appends the contents of seq to list |
| 4 | [list.index(obj)](https://www.tutorialspoint.com/python/list_index.htm)  Returns the lowest index in list that obj appears |
| 5 | [list.insert(index, obj)](https://www.tutorialspoint.com/python/list_insert.htm)  Inserts object obj into list at offset index |
| 6 | [list.pop(obj=list[-1])](https://www.tutorialspoint.com/python/list_pop.htm)  Removes and returns last object or obj from list |
| 7 | [list.remove(obj)](https://www.tutorialspoint.com/python/list_remove.htm)  Removes object obj from list |
| 8 | [list.reverse()](https://www.tutorialspoint.com/python/list_reverse.htm)  Reverses objects of list in place |
| 9 | [list.sort([func])](https://www.tutorialspoint.com/python/list_sort.htm)  Sorts objects of list, use compare func if given |

**CHAPTER4 :TUPLES**

* A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple

thistuple = ("apple", "banana", "cherry")  
print(thistuple)

* Basically almost everything that we have applied to a list can also be applied to a tuple example selecting a tuple using a range eg

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[2:5])

* Never the less we there are few things which cannot be applied to a tuple directly

Example

Change Tuple Values

x = ("apple", "banana", "cherry")  
y = list(x)  
y[1] = "kiwi"  
x = tuple(y)  
  
print(x)

Remove Items

* Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely

Example

thistuple = ("apple", "banana", "cherry")  
del thistuple  
print(thistuple) #this will raise an error because the tuple no longer exist

Basic Tuples Operations

* Tuples respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

|  |  |  |
| --- | --- | --- |
| **Python Expression** | **Results** | **Description** |
| len((1, 2, 3)) | 3 | Length |
| (1, 2, 3) + (4, 5, 6) | (1, 2, 3, 4, 5, 6) | Concatenation |
| ('Hi!',) \* 4 | ('Hi!', 'Hi!', 'Hi!', 'Hi!') | Repetition |
| 3 in (1, 2, 3) | True | Membership |
| for x in (1, 2, 3): print x, | 1 2 3 | Iteration |

Indexing, Slicing, and Matrixes

Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings. Assuming following input −

L = ('spam', 'Spam', 'SPAM!')

|  |  |  |
| --- | --- | --- |
| **Python Expression** | **Results** | **Description** |
| L[2] | 'SPAM!' | Offsets start at zero |
| L[-2] | 'Spam' | Negative: count from the right |
| L[1:] | ['Spam', 'SPAM!'] | Slicing fetches sections |

Built-in Tuple Functions

Python includes the following tuple functions −

|  |  |
| --- | --- |
| **Sr.No.** | **Function with Description** |
| 1 | [cmp(tuple1, tuple2)](https://www.tutorialspoint.com/python/tuple_cmp.htm)  Compares elements of both tuples. |
| 2 | [len(tuple)](https://www.tutorialspoint.com/python/tuple_len.htm)  Gives the total length of the tuple. |
| 3 | [max(tuple)](https://www.tutorialspoint.com/python/tuple_max.htm)  Returns item from the tuple with max value. |
| 4 | [min(tuple)](https://www.tutorialspoint.com/python/tuple_min.htm)  Returns item from the tuple with min value. |
| 5 | [tuple(seq)](https://www.tutorialspoint.com/python/tuple_tuple.htm)  Converts a list into tuple. |

**CHAPTER5:Dictionaries**

* A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

### Example

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict)

Accessing Items

* There is a method called get() that will give you the same result:

Example

x = thisdict.get("model")

Change Values

Example

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["year"] = 2018

Loop Through a Dictionary

* When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

Example

for x in thisdict:  
  print(x)

### Example

* Print all values in the dictionary, one by one:

for x in thisdict:  
  print(thisdict[x])

Check if Key Exists

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
if "model" in thisdict:  
  print("Yes, 'model' is one of the keys in the thisdict dictionary")

Nested Dictionaries

myfamily = {  
  "child1" : {  
    "name" : "Emil",  
    "year" : 2004  
  },  
  "child2" : {  
    "name" : "Tobias",  
    "year" : 2007  
  },  
  "child3" : {  
    "name" : "Linus",  
    "year" : 2011  
  }  
}

Built-in Dictionary Functions & Methods

|  |  |
| --- | --- |
| **Sr.No.** | **Function with Description** |
| 1 | [cmp(dict1, dict2)](https://www.tutorialspoint.com/python/dictionary_cmp.htm)  Compares elements of both dict. |
| 2 | [len(dict)](https://www.tutorialspoint.com/python/dictionary_len.htm)  Gives the total length of the dictionary. This would be equal to the number of items in the dictionary. |
| 3 | [str(dict)](https://www.tutorialspoint.com/python/dictionary_str.htm)  Produces a printable string representation of a dictionary |
| 4 | [type(variable)](https://www.tutorialspoint.com/python/dictionary_type.htm)  Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type. |

Python includes following dictionary methods −

|  |  |
| --- | --- |
| **Sr.No.** | **Methods with Description** |
| 1 | [dict.clear()](https://www.tutorialspoint.com/python/dictionary_clear.htm)  Removes all elements of dictionary *dict* |
| 2 | [dict.copy()](https://www.tutorialspoint.com/python/dictionary_copy.htm)  Returns a shallow copy of dictionary *dict* |
| 3 | [dict.fromkeys()](https://www.tutorialspoint.com/python/dictionary_fromkeys.htm)  Create a new dictionary with keys from seq and values *set* to *value*. |
| 4 | [dict.get(key, default=None)](https://www.tutorialspoint.com/python/dictionary_get.htm)  For *key* key, returns value or default if key not in dictionary |
| 5 | [dict.has\_key(key)](https://www.tutorialspoint.com/python/dictionary_has_key.htm)  Returns *true* if key in dictionary *dict*, *false* otherwise |
| 6 | [dict.items()](https://www.tutorialspoint.com/python/dictionary_items.htm)  Returns a list of *dict*'s (key, value) tuple pairs |
| 7 | [dict.keys()](https://www.tutorialspoint.com/python/dictionary_keys.htm)  Returns list of dictionary dict's keys |
| 8 | [dict.setdefault(key, default=None)](https://www.tutorialspoint.com/python/dictionary_setdefault.htm)  Similar to get(), but will set dict[key]=default if *key* is not already in dict |
| 9 | [dict.update(dict2)](https://www.tutorialspoint.com/python/dictionary_update.htm)  Adds dictionary *dict2*'s key-values pairs to *dict* |
| 10 | [dict.values()](https://www.tutorialspoint.com/python/dictionary_values.htm)  Returns list of dictionary *dict*'s values |