**5. DataTypes Introduction**

* 1. Numbers Page 1
  2. String Page7
  3. List Page18
  4. Tuple Page 22
  5. Dictionary Page 25
  6. Set Page 31

Every value in Python has a **datatype**.

Type represents the kind of value and determines how the value can be used. Everything in Python is an object and every object has an identity, a type, and a value.

Like another object-oriented language such as Java or C++, there are several data types which are built into Python. Extension modules which are written in C, Java, or other languages can define additional types.

To determine a variable's type in Python you can use the **type()** function.

**Sequences :**

**There are 6 sequence types**

1. **Strings name = “Python” 0 1 2 3 4 5**
2. **Lists emp\_ids = [1,2,3,4,5,6]**
3. **Tuples emp\_ids = (1,2,3,4,5,6)**
4. Bytearrays
5. buffers and
6. **range objects**

for i in *"Python Programming"*:

print(i)

print(*"----------------"*)

for i in [4,3,1,5,7,34,1]:

print(i)

print(*"----------------"*)

for i in (1,3,5,64,2,32,1,3,5,6,7):

print(i)

print(*"----------------"*)

for i in range(10):

print(i)

**CRUD operations**

**Create – Create data/table….. C R U D**

**Retrieve – Select data**

**Update - Modify data**

**Delete – Delete data**

There are certain things you can do with **all sequence types**.

These operations include

**1. indexing R**

1. **slicing R**
2. **adding U**
3. **multiplying and R**
4. checking for **membership**. **R**

In addition, Python has **built-in functions** for len() max() min()

**finding the length of a sequence** and

**for finding its largest and smallest elements.**

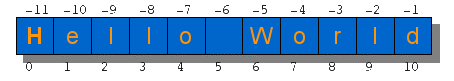
**9.2 String:**

Strings are amongst the most popular types in Python. We can create them simply by **enclosing characters** in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable.

For example –

-3 -2 -1 0 1 2 3 4

Var1 = "Hello World” var2 = “Python programming”



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |

……

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |

7 6 5 4 3 2 1 0

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| H | E | L | L | 0 | [ | 1 | , | 2 |

## message = “Hello World”

## 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0

## 2 2 2 2 2 2 2 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |

## 0 123456 1 2

## 123456

## Indexing(Accessing Values) in Strings:

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for **slicing** along with the index or indices to obtain your substring.

For example –

message1='Hello World'

message2="Python Programming"

print("var1[0]: ", var1[0])

message1[0]: H

message2[1:5]: ytho

## Slicing(Accessing Values) in Strings:

## We can also call out a range of characters from the string. Say we would like to just print the word World. We can do so by creating a slice, which is a sequence of characters within an original string. With slices, we can call multiple character values by creating a range of index numbers separated by a colon [x:y]

## Print(message1[6:12])

## 

## X = 10

## 4567 X

## Y = 10

## Y

## del X

## A2+B2 Y2+5Y2

## X+X

## 2x

## X2+X2 = 2x2

## X+4X = 5X

## Q)X2+2X+Y=10. WHEN X IS 3 FIND THE VALUE OF Y.

## As given x=3

## 32+2(3)+Y = 10

## 9+6+Y = 10

## 15+Y = 10

## Y = 10-15

## Y = -5

## 10+”World” “10”+”World” 10World

## CRD - No update for immutable data(Numbers,String)

## var1 = “Hello”

## var1 12345

## var1 = var1 + “ World”

## 54678

## Updating Strings

## You can "update" an existing string by (re)assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether.

## For example −

var1 ='Hello World'

var1 = var1 + ” Welcome to Python”

Updated String :- Hello World Python

## var1

Hello World

Hello World Wecome to python

## variable can refer single memory location

## var1 = “Hello World”

## var1 = “Hello Wolrld.Welcome to Python”

## memory location addr can be referred by multiple variables

## a = b = c = 1

## a

## b

1

## c

## Built-in String Methods:

## num = “10”

## print( type( int(num+”20”) ) )

Python includes the following built-in methods to manipulate strings −

|  |  |
| --- | --- |
| **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. |

## 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:

One of Python's coolest features is the string format operator **%**. This operator is unique to strings and makes up for the pack of having functions from C's printf() family. Following is a simple example −

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

My name is Zara and weight is 21 kg!

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 | unsigned 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 |

## Triple Quotes:

Python's triple quotes comes to the rescue by allowing strings to span multiple lines, including verbatim NEWLINEs, TABs, and any other special characters.

The syntax for triple quotes consists of **three consecutive single or doublequotes.**

para\_str ="""this is a long string that is made up of

several lines and non-printable characters such as

TAB ( \t ) and they will show up that way when displayed.

NEWLINEs within the string, whether explicitly given like

this within the brackets [ \n ], or just a NEWLINE within

the variable assignment will also show up.

"""

print para\_str

this is a long string that is made up of

several lines and non-printable characters such as

TAB ( ) and they will show up that way when displayed.

NEWLINEs within the string, whether explicitly given like

this within the brackets [

], or just a NEWLINE within

the variable assignment will also show up.

print'C:\\nowhere'

C:\nowhere

print'C:\\nowhere'

C:\\nowhere

## Unicode String:

Normal strings in Python are stored internally as 8-bit(1 byte) ASCII, while Unicode strings are stored as 16-bit(2 bytes) Unicode. This allows for a more varied set of characters, including special characters from most languages in the world. I'll restrict my treatment of Unicode strings to the following –

print u'Hello, world!'

Hello, world!