Python (3.8 Dev) dated: 12-aug-2018

Python is a very high level language.

-- It has high level built in data types such as, flexible arrays and dictionaries.

Python allows us to split our program into modules that can be reused in other pyhton programs.

It is an interpreted language.

--high level data types allow us to express complex operations in a single statement.

--statement grouping is done by indentation instead of beginning & end brackets.

--no variable or argument declarations are necessary.

Python is extensible.

Operators:

+ - \* / % =

Division operator ( / ) always returns a float. To do floor division and get an integer result (discarding any fractional result) we can use the ( // ) operator.

# --comment

**\*\* --calculate powers**

eg., 2 \*\* 7 # 2 to the power of 7

If a variable is “not defined” (assigned a value), trying to use it will give us an error.

eg.,

>>> n # try to access an undefined variable

Traceback (most recent call last):

File “<stdin>”, Line 1, in <module>

NameError: name ‘n’ is not defined

In interactive mode:

**\_ # underscore, the last printed expression is assigned to the variable ‘ \_ ’**

Data types:

Number: int, float, decimal, fraction, complex numbers.

Strings: “ string literal“ or ‘ string literal‘

\n # new line

print() #writes output

\ # to escape quote

eg.,

name=”www \”world wide web\””

**Raw Strings: If we don’t want characters prefaced by ‘ \ ‘ to be interpreted as special chars, we can use raw string by adding an ‘ r ‘ before the first quote;**

eg.,

>>> print (‘c:\some\name’) # \n means new line

output: [c:\some](../../../../c:/some)

ame

>>> print(r’c:\some\name’)

output: c:\some\name

**Multiple lines string literlas ( “ “ “ ) or ( ‘ ‘ ‘ ) three quotes**

eg.,

>>> print(“””

... danish wani

... buchpora

... 25

... “””)

output: danish wani

buchpora

25

**‘ + ‘ #concatenation operator**

eg.,

firstname=”danish”

lastname=”wani”

print (firstname+lastname)

output: danishwani

**‘ \* ‘ # strings can be repeated using ‘ \* ‘ operator**

eg.,

>>> 3 \* ‘un’ + ‘ium’

output: unununium

two or more sting literals next to each other are automatically concatenated.

>>> “pyt””hon”

output: pyhton

# works with two literals though, not with variables or expressions

If we want to concatenate variables or a variable & a literal, we use ‘ + ‘

eg..,

>>> prefix=’py’

prefix+’thon’

output: ‘python’

--Strings can be indexed, with the first character having index 0.

There is no separate character type; a character is simply a string of size one:

>>> word=”python”

word[0] #char in position 0

output:

‘p’

--Indices may also be negative numbers, to start counting from right.

eg.,

>>> word[-1]

output: ‘n’

**Slicing**: to obtain substring

>>> word[0:2]

output: ‘py’

syntax: [start:end] #start is always included & end is always excluded

**s[:i] + s[i:] #always gives ‘s’**

D A N I S H

INDICES: 0 1 2 3 4 5

-6 -5 -4 -3 -2 -1

Attempting to use an index that is too large will result in an error:

IndexError: string index out of range

**Python strings cannot be changed --- they are immutable**

>>> name[0]=’t’

TypeError: ‘str’ object does not support item assignment.

**Length of a string len(stringi)**

eg.,

>>> len(name)

output: 6