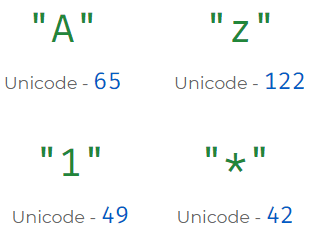
**Comparing Strings**

Computer internally stores characters as numbers.

Every character has a unique **Unicode** value.



Ord

To find the Unicode value of a character, we use the

ord()

ord(character)

gives unicode value of the character.

**Code**



1

2

unicode\_value = ord("A")

print(unicode\_value)

PYTHON

**Output**



65

chr

To find the character with the given Unicode value, we use the

chr()

chr(unicode)

gives character with the unicode value.

**Code**



1

2

char = chr(75)

print(char)

PYTHON

**Output**



K

Unicode Ranges

*48 - 57* -> Number Digits (0 - 9)

*65 - 90* -> Capital Letters (A - Z)

*97 - 122* -> Small Letters (a - z)

*Rest* -> Special Characters, Other Languages

Printing Characters

The below code will print the characters from

A

to

Z

**Code**



1

2

for unicode\_value in range(65,91):

print(chr(unicode\_value))

PYTHON

**Output**



A

B

C

D

E

F

G

H

I

J

K

Comparing Strings

In Python, strings are compared considering unicode.

**Code**



1

print("A" < "B")

PYTHON

**Output**



True

As unicode value of

A

is 65 and

B

is 66, which internally compares

65 < 66

. So the output should be

True

Character by Character Comparison

In Python, String Comparison is done character by character.

**Code**



1

print("BAD" >= "BAT")

PYTHON

**Output**



False

**Code**



1

print("98" < "984")

PYTHON

**Output**



True

**Best Practices**

Naming Variables Rule #1

Use only the below characters

* Capital Letters ( A – Z )
* Small Letters ( a – z )
* Digits ( 0 – 9 )
* Underscore(\_)

*Examples:*  
age, total\_bill

Naming Variables Rule #2

Below characters cannot be used

* Blanks ( )
* Commas ( , )
* Special Characters  
  ( ~ ! @ # $ % ^ . ?, etc. )

Naming Variables Rule #3

Variable name must begin with

* Capital Letters ( A – Z )
* Small Letters ( a – z )
* Underscore( \_ )

Naming Variables Rule #4

Cannot use Keywords, which are reserved for special meaning

* int
* str
* print

 etc.,

**Keywords**

Words which are reserved for special meaning

**Code**



1

help("keywords")

PYTHON

**Output**



Here is a list of the Python keywords. Enter any keyword to get more help.

False break for not

None class from or

True continue global pass

\_\_peg\_parser\_\_ def if raise

and del import return

as elif in try

assert else is while

async except lambda with

await finally nonlocal yield

Case Styles

* Camel case: **totalBill**
* Pascal case: **TotalBill**
* Snake case: **total\_bill**

Snake case is preferred for naming the variables in Python.

Notes

**Round**

Rounding Numbers

round(number, digits(optional))

Rounds the float value to the given number of decimal digits.

digits

-> define the number of decimal digits to be considered for rounding.

* when not specified default is

0

**Code**



1

2

3

4

a = round(3.14,1)

print(a)

a = round(3.14)

print(a)

PYTHON

**Output**



3.1

3

Floating Point Approximation

Float values are stored approximately.

**Code**

print(0.1 + 0.2)

**Output**



0.30000000000000004

Floating Point Errors

Sometimes, floating point approximation gives unexpected results.

**Code**



1

print((0.1 + 0.2) == 0.3)

PYTHON

**Output**



False

To avoid these unexpected results, we can use

round()

**Code**



1

2

3

a = round((0.1 + 0.2), 1)

print(a)

print(a == 0.3)

PYTHON

**Output**



0.3

True

Comments

Comment starts with a hash

#

It can be written in its own line next to a statement of code.

**Code**



1

2

3

4

n = 5

# Finding if Even

even = (n % 2 == 0)

print(even) # prints boolean value

PYTHON

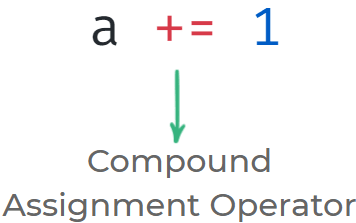
**Output**



False

Notes

**Compound Assignment Operators**



Different compound assignment operators are

+=

,

-=

,

\*=

,

/=

,

%=

a += 1

is similar to

a = a + 1

**Code**



1

2

3

a = 10

a -= 2

print(a)

PYTHON

**Output**



8

**Examples of Compound Assignment Operators**

**Code**



1

2

3

a = 10

a /= 2

print(a)

PYTHON

**Output**



5.0

**Code**



1

2

3

a = 10

a %= 2

print(a)

PYTHON

**Output**



0

**Escape Characters**

Single And Double Quotes

String is a sequence of characters enclosed within quotes.

**Code**



1

2

3

4

sport = 'Cricket'

print(type(sport))

sport = "Cricket"

print(type(sport))

PYTHON

**Output**



<class 'str'>

<class 'str'>

**Code**



1

2

is\_same = ('Cricket' == "Cricket")

print(is\_same)

PYTHON

**Output**

True

Escape Characters

Escape Characters are a sequence of characters in a string that are interpreted differently by the computer. We use escape characters to insert characters that are illegal in a string.

**Code**



1

print("Hello\nWorld")

PYTHON

**Output**



Hello

World

We got a new line by adding

\n

escape character.

Examples - Escape Characters

Escape Characters start with a backslash in Python

* \n

 -> New Line

* \t

 -> Tab Space

* \\

 -> Backslash

* \'

 -> Single Quote

* \"

 -> Double Quote

Passing Strings With Quotes

The backslash

\

character here tells Python not to consider the next character as the ending of the string.

**Code**



1

print('It\'s Python')

PYTHON

**Output**



It's Python

**Code**



1

print("It's Python")

PYTHON

**Output**



It's Python

Notes

Dis