**STRINGHE**

Ricordiamo che le stringhe sono iterabili IMMUTABILI. L’iteratore itera (che brutto italiano!) la stringa carattere per carattere. Vediamo un esempio.

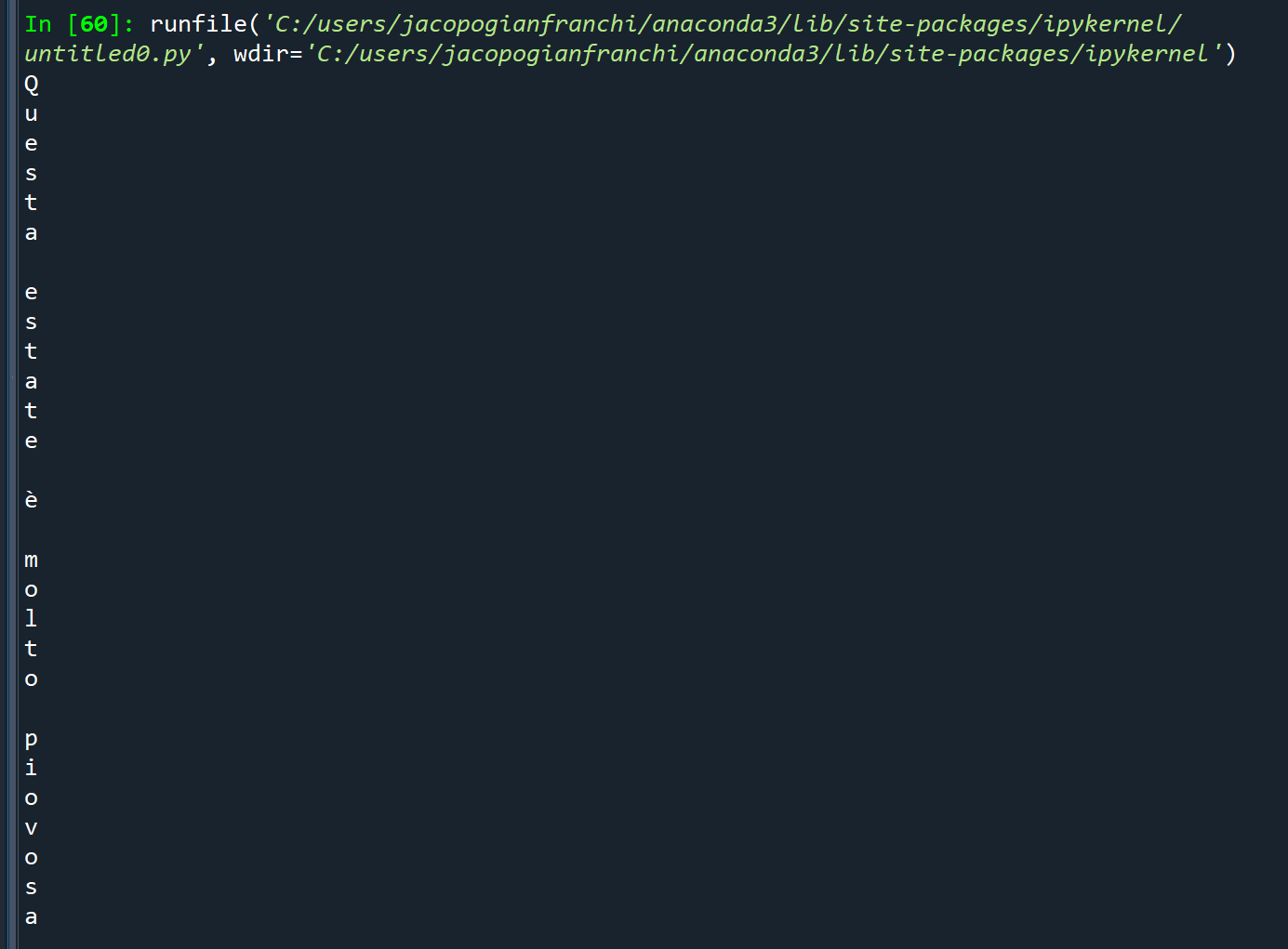
Codice Python:

frase = 'Questa estate è molto piovosa.'

for x in frase:

    print(x)

Console:



Le stringhe possono essere inserite all’interno del singolo apice, dei doppi apici o dei tripli apici.

Di solito si inseriscono tra singoli apici. Per inserire i simboli all’interno di una stringa contenuta tra singoli apici, è necessario utilizzare i caratteri di escape.

Come detto all’inizio del capitolo, le stringhe sono immutabili 🡺 Non è possibile modificarle in nessun modo.

**METODO len()**

Il metodo len calcola il numero di elementi presenti in una stringa. Vediamo qualche esempio.

Codice Python:

frase1 = 'Questa estate è molto piovosa.'

frase2 = 'Questa estate ha rotto le palle.'

frase3 = 'Questa estate è l\'unica che abbia davvero rotto le palle'

frase4 = 'L\'estate è formata da: 1.GIUGNO\t2.LUGLIO\t3.AGOSTO.'

print(frase1,len(frase1))

print()

print(frase2,len(frase2))

print()

print(frase3,len(frase3))

print()

print(frase4,len(frase4))

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Dal precedente screen si evince che i caratteri di escape vengono considerati come unico carattere.

Vediamo un caso un po particolare. Consideriamo la seguente stringa.

Codice Python

canzone = '''Dove seeeei

l'estate comincia adessooo

C'è l'apocalisse in centrooo'''

print(canzone, len(canzone))

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Contando i caratteri otteniamo il numero 65. Quindi perché la funzione len ritorna 67?

Questo perche, quando si va a capo, è sottointeso il carattere di escape \n. Essendo un carattere di escape, nonostante i simboli siano 2 (\ ed n) lo consideriamo come unico carattere.

Poiché si va a capo due volte abbiamo due \n da contare 🡺 65+2=67.

**SUBSETTING**

Il subsetting per le stringhe funziona esattamente come nel caso delle liste. In subset estrae un sottoinsieme di caratteri da una stringa, e lo inserisce in una nuova variabile (O lo ritorna direttamente). Il subsetting, quindi, non modifica la stringa di partenza. Ricordiamo che le stringhe sono immutabili 🡺 Non è possibile modificare la stringa di partenza in nessun caso!

Vediamo alcuni esempi.

Codice Python:

frase = 'Questa estate è molto piovosa.'

# Estrazione dall'elemento 2 incluso all'elemento 8 escluso:

print(frase[2:8]) # esta e

# Estrazione dall'elemento 16 fino alla fine della stringa:

print(frase[16:])  # molto piovosa.

# Estrazione fino all'elemento 9 escluso

print(frase[:9])   # Questa es

Console:

Immagine che contiene schermata, testo, Carattere

Descrizione generata automaticamente

**METODI**

Proviamo a vedere quali metodi possono essere applicati alle stringhe tramite la funzione dir.

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Studiamo quindi i metodi che compariranno all’esame

**ORD() e CHR()**

Sappiamo che i caratteri scritti nello script vengono convertiti da python in numeri ASCII. Il metodo ord() accetta un char come input, e ritorna il corrispettivo valore ascii. Il metodo chr() è l’inverso del metodo ord(). Accetta un numero ascii e lo converte nel char corrispondente.

Riportiamo qui sotto la tabella dei codici ascii:

TABELLA CODICI ASCII

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Vediamo qualche esempio con le funzioni ord() e chr().

**ORD()**

Codice Python:

print(ord("A"))

print()

print(ord("%"))

print()

print(ord("q"))

print()

print(ord("8"))

print()

print(ord("N"))

print()

print(ord("\n"))

print()

print(ord("l"))

print()

Console:

Immagine che contiene testo, schermata

Descrizione generata automaticamente

­**CHR()**

Codice Python:

print(chr(65))

print()

print(chr(37))

print()

print(chr(113))

print()

print(chr(56))

print()

print(chr(78))

print()

print(chr(10))

print()

print(chr(108))

print()

Console:

Immagine che contiene testo, schermata

Descrizione generata automaticamente

**CONCATENAZIONE STRINGHE**

E’ possibile concatenare due stringhe utilizzando il simbolo +. La concatenazione di due stringhe permette di creare una nuova stringa contenete le stringhe precedenti unite (joint).

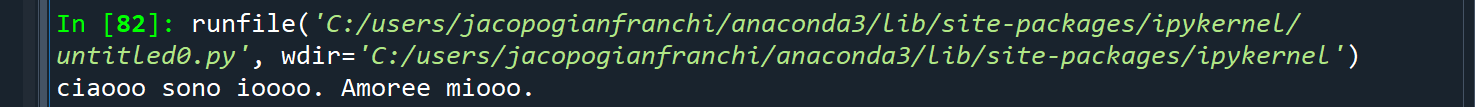
Vediamo qualche esempio.

Codice Python:

stringa\_concatenata = "ciaooo" + " sono" + " ioooo." + " Amoree" + " miooo."

print(stringa\_concatenata)

Console:



**RIPETIZIONE STRINGHE**

E’ possibile anche ripetere più volte una stringa, “moltiplicandola” per un intero.

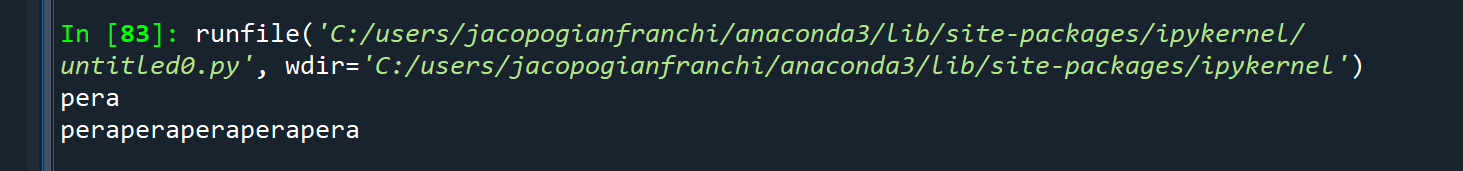
Codice Python:

frutto = "pera"

print(frutto)

print(5\*frutto)

Console:



Ovviamente, non è possibile “moltiplicare” una stringa per un numero != int. Se ci provassimo, otterremo il seguente errore:

Codice Python:

frutto = "pera"

print(frutto)

print(2.5\*frutto)

Console:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

Poiché le stringhe sono immutabili, non è possibile eliminare un elemento al loro interno. Ricordiamo la keyword **del** utilizzata per le liste:

Codice Python:

frutta = ["mela","pera","banana","noce","albicocca","pesca"]

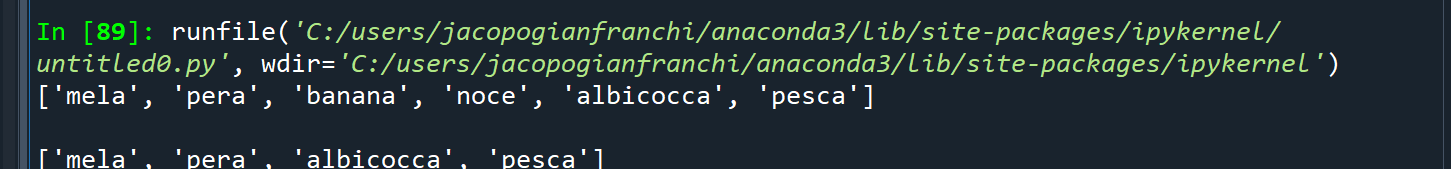
print(frutta)

print()

del frutta[2:4]

print(frutta)

Console:



La keyword del ha modificato la lista di partenza 🡺 da questo già si capisce che per le stringhe non può funzionare, perché esse sono immutabili.

Vediamo un’ultima cosa.

Se non utilizzassimo il subsetting, si nota che la keyword del elimina tutta la lista

Codice Python:

frutta = ["mela","pera","banana","noce","albicocca","pesca"]

print(frutta)

print()

del frutta

print(frutta)

Console:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

Ok.

Per le stringhe, quindi, varrà quanto segue:

* La keyword del con il subsetting non funzionerà
* La keyword del applicata a tutta la lista ne causerà l’eliminazione.

Vediamo con il solito esempio.

Codice Python:

frutto = "Melograno"

print(frutto)

print()

del frutto

print(frutto)

Console:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

Ovviamente.

Proviamo ad eliminare tutta la stringa:

Codice Python:

frutto = "Melograno"

print(frutto)

print()

del frutto

print(frutto)

Console:

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente

Ok

**MIN() E MAX()**

I metodi min e max ritornano, rispettivamente, il carattere più piccolo ed il carattere più grande di una stringa, in base al loro valore ascii. Vediamo qualche esempio.

Codice Python:

frase = "Ho tanta voglia di salame"

print(min(frase))

print(ord(min(frase)))

print(max(frase))

print(ord(max(frase)))

Console:

Immagine che contiene schermata, testo, Carattere

Descrizione generata automaticamente

Quanto visto nel nostro esempio mostra ciò che vediamo nella tabella a pagina 6. Per esercizio trasformare le lettere componenti la nostra frase in ascii, ed estrarre minimo e massimo.

Codice Python:

frase = "Ho tanta voglia di salame"

print(frase)

print()

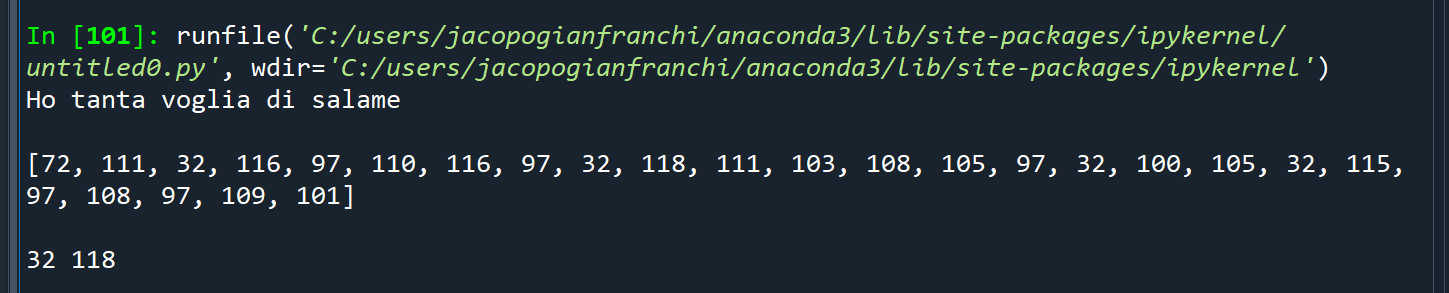
frase\_ascii = [ord(i) for i in frase]

print(frase\_ascii)

print()

print(min(frase\_ascii),max(frase\_ascii))

Console:



Quindi, dalla trasformazione da char ad ascii si vede che il numero più basso assume valore 32, ed è il numero relativo al carattere spazio, mentre il numero più alto assume valore 118, e corrisponde al carattere ‘v’.

**INDEX()**

Il metodo index() ritorna la posizione di un carattere definito dall’utente in una stringa. Se il carattere è ripetuto più volte, tale funzione ritorna la prima posizione che trova nel quale si trova la stringa.

Questo metodo può essere riassunto nel seguente modo:

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Vediamo un esempio.

Codice Python:

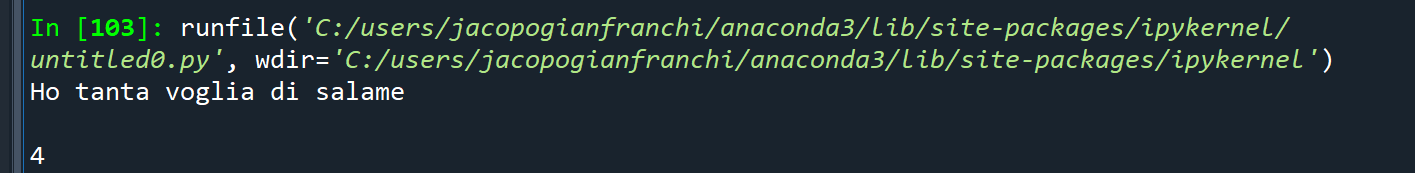
frase = "Ho tanta voglia di salame"

print(frase)

print()

print(frase.index("a"))

Console:



Il carattere ‘a’ si incontra per la prima volta in posizione 4 nella stringa frase. La funzione index() ritorna quindi 4. Si noti che il carattere ‘a’ si trova anche in altre posizioni: 7 e 14. Tuttavia, la funzione index() ritorna solo la prima posizione.

Abbiamo finora utilizzato la funzione index per trovare un solo carattere. Utilizziamola ora per trovare più di un carattere.

Codice Python:

frase = "Ho tanta voglia di salame"

print(frase)

print()

print(frase.index("ta"))

Console:  
Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Notiamo che la sottostringa “ta” compare due volte nella stringa. Ovviamente il metodo index prenderà in considerazione per prima cosa la prima volta. Si noti che la ‘t’ si trova in posizione 3, la ‘a’ si trova in posizione 4. Ovviamente il metodo index() prenderà in considerazione la posizione del primo carattere, ossia 3. 3 è proprio il valore di ritorno.

Infine, si noti che se cerchiamo un carattere che non esiste nella stringa, il metodo indexI() ritornerà errore.

Codice Python:

frase = "Ho tanta voglia di salame"

print(frase)

print()

print(frase.index("q"))

Console:  
Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente

**FIND()**

Il metodo find cerca una sottostringa come avviene per il metodo index. Tuttavia, nel caso in cui non la trova, non restituisce errore ma restituisce -1:

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Vediamo qui sotto qualche esempio.

Codice Python:

frase = "Ho tanta voglia di salame"

print(frase)

print()

print(frase.find("a"))    # 4

print()

print(frase.find("q"))    # -1

print()

print(frase.find("ta"))   #3

print()

print(frase.find("me")) # 4

Console:

Immagine che contiene testo, schermata

Descrizione generata automaticamente

.

Facciamo ora una prova con Python. Supponiamo di voler cercare la posizione nella quale si trova la lettera a per la terza volta nella stringa.

Codice Python:

frase = "Ho tanta voglia di salame"

print(frase)

print()

print(frase.find("a",10))    # 14

Console:

Immagine che contiene schermata, testo, Carattere

Descrizione generata automaticamente

Vediamo infine come si utilizza il terzo paramentro. Supponiamo di voler cercare la lettera ‘a’ dalla posizione 14 alla posizione 17. Cosa avremo come valore di ritorno?

Codice Python:

frase = "Ho tanta voglia di salame"

print(frase)

print()

print(frase.find("a",14,17))    # 14

Console:

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Quanto visto mostra che il limite inferiore è preso in considerazione.

Consideriamo, infine, un esempio. Così come avviene per lo slicing, il limite superiore del metdoo find viene escluso, mentre il limite inferiore viene preso in considerazione. Vediamo un esempio.

Codice Python:

frase = "Ho tanta voglia di salame"

print(frase.find("s",19,22))    #19

# Il programma cercherà la lettera s partendo dalla posizione 19.

# La lettera s si troverà proprio nella posizione di partenza 19 ==>

# ritornerà il numero 19

print(frase.find("s",10,19))   # -1

# Il programma cercherà la lettera s partendo dalla posizione 10 e cercherà

# In tutta la stringa fino alla lettera 19. Quando arriverà a 19 si arresterà.

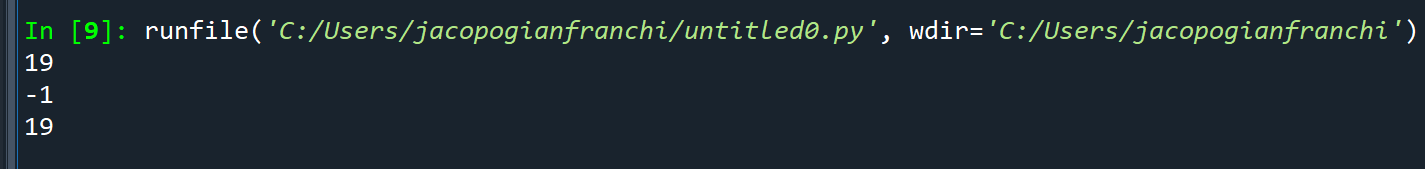
# Ma Python non considera l'estremo superiore, quindi si ferma a 18. A 18 non trova

# nulla ==> Ritorna -1. Per trovare la lettera s bisogna quindi cercare fino alla

# posizione 20, in questo modo si arresterà a 19 ==> Ritornerà 19.

print(frase.find("s",10,20))   # 19    OK

Console:



**ALGORITMO PER TROVARE LE TUTTE LE POSIZIONI DI UN CARATTERE**

Codice Python:

frase = "Ho tanta voglia di salame"

print(frase)

print()

a1 = frase.find("a")        # 4

print(a1)

a2 = frase.find("a",a1+1)   # 7

print(a2)

a3 = frase.find("a",a2+1)  # 14

print(a3)

a4 = frase.find("a",a3+1)  # 20

print(a4)

Console:

**Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente**

**RFIND()**

Rfind è l’esatto opposto di find. Trova l’ultima posizione in cui si elemento desiderato.

Anche rfind accetta tre argomenti:

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Vediamo qualche esempio.

Codice Python:

frase = "Ho tanta voglia di salame"

print(frase)

print()

a1 = frase.rfind("a")         # 22

print(a1)

a2 = frase.rfind("a",0,a1-1)  # 20

print(a2)

a3 = frase.rfind("a",0,a2-1)  # 14

print(a3)

# E così via

Console:

Immagine che contiene testo, schermata

Descrizione generata automaticamente

**ISALNUM()**

Il metodo isalnum ritorna True se la stringa considerata è composta da lettere, numeri, oppure numeri e lettere. Se nella stringa sono contenuti simboli, come lo spazio, %,£,$,%,/ o un qualsiasi altro simbolo, tale funzione ritornerà False.

Immagine che contiene testo, schermata, linea, Carattere

Descrizione generata automaticamente

Vediamo qualche esempio.

Codice Python:

frase = "Che pizza"

print(frase.isalnum())

frase2 = "15"

print(frase2.isalnum())

frase3 = "OtTonE"

print(frase3.isalnum())

frase4 = "ViaValDiCogne22"

print(frase4.isalnum())

frase5 = "Via Val Di Cogne 22"

print(frase5.isalnum())

frase6 = "16/10/1958"

print(frase6.isalnum())

frase7 = "Holallà"

print(frase7.isalnum())

frase8 = "L'autobus"

print(frase8.isalnum())

Console:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

**ISALPHA()**

Molto simile al metodo precedente, ossia isalnum(), ma ritorna True solo se tutti gli elementi della stringa sono lettere. In caso di presenza di un numero, uno spazio o un qualsiasi simbolo, la funzione isAlpha ritorner False.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Rivediamo gli esempi appena visti in precedenza sostituendo isalnum() con isalpha():

Codice Python:

frase = "Che pizza"

print(frase.isalpha())

frase2 = "15"

print(frase2.isalpha())

frase3 = "OtTonE"

print(frase3.isalpha())

frase4 = "ViaValDiCogne22"

print(frase4.isalpha())

frase5 = "Via Val Di Cogne 22"

print(frase5.isalpha())

frase6 = "16/10/1958"

print(frase6.isalpha())

frase7 = "Holallà"

print(frase7.isalpha())

frase8 = "L'autobus"

print(frase8.isalpha())

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**ISDIGIT()**

Il metodo isdigit() ritorna True se tutti gli elementi della stringa sono cifre. Vengono considerati cifre anche gli esponenti.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Vediamo gli esempi.

Codice Python:

frase = "Che pizza"

print(frase.isdigit())

frase2 = "15"

print(frase2.isdigit())

frase3 = "OtTonE"

print(frase3.isdigit())

frase4 = "ViaValDiCogne22"

print(frase4.isdigit())

frase5 = "Via Val Di Cogne 22"

print(frase5.isdigit())

frase6 = "16/10/1958"

print(frase6.isdigit())

frase7 = "Holallà"

print(frase7.isdigit())

frase8 = "L'autobus"

print(frase8.isdigit())

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**ISLOWER()**

Islower() ritorna True se all’interno della stringa si ha almeno un carattere alfabetico, e se i caratteri alfabetici contenuti in esso devono essere tutti minuscoli. Per il resto, all’interno della stringa, possono esserci numeri, spazi, simboli e altro.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Codice Python:

frase = "che pizza margherita"

print(frase.islower())

frase2 = "15 f"

print(frase2.islower())

frase3 = "ottone "

print(frase3.islower())

frase4 = "ViaValDiCogne22"

print(frase4.islower())

frase5 = "Via Val Di Cogne 22"

print(frase5.islower())

frase6 = "16/10/1958"

print(frase6.islower())

frase7 = "hollallà"

print(frase7.islower())

frase8 = "L'autobus"

print(frase8.islower())

Console:

Immagine che contiene testo, Software multimediale, software, Software per la grafica

Descrizione generata automaticamente

**ISUPPER()**

Funzione opposta ad islower().

**ISSPACE()**

**Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente**

Codice Python:

frase = "che pizza margherita"

print(frase.isspace())

frase2 = "15 f"

print(frase2.isspace())

frase3 = "ottone "

print(frase3.isspace())

frase4 = "   "

print(frase4.isspace())

frase5 = "Via Val Di Cogne 22"

print(frase5.isspace())

frase6 = "16/10/1958"

print(frase6.isspace())

frase7 = "hollallà"

print(frase7.isspace())

frase8 = "L'autobus"

print(frase8.isspace())

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**JOIN()**

Finalmente si torna a studiare metodi decenti. Il metodo Join unisce tra loro due o più stringhe. Esso accetta un solo argomento, ed esso è un iterabile (lista,tupla etc) i cui elementi sono le stringhe che si voglionoi concatenare. Il valore di ritorno sarà una stringa

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Vediamo qualche esempio.

Codice Python:

nome = "Ascanio"

cognome = "De Ascanis"

separatore = " "

concatenazione = separatore.join((nome,cognome))

print(concatenazione)

# E' anche comodo evitare di inserire tutto in una variabile:

print()

print(" ".join(["Mario","De Cretiniis"]))

print()

print(" ".join(["Giada","Maria","Manuela","De Sederiis"]))

# CAMBIAMO IL SEPARATORE

print()

print("???".join(["Flavia","Ilaria","De Scemis"]))

print()

print("asK".join(["Crisso","Spartacus","Deficientis"]))

Console:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

**SPLIT()**

Il metodo split separa le stringhe quando trova un separatore. Il separatore può essere definito dall’utente, oppure, di default, assume il valore ‘ ‘. Il valore di ritorno del metodo split è una lista.

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Vediamo degli esempi.

Codice Python:

# Non si passano parametri. In questo caso, si divide la stringa in base allo # spazio

print("Hai rotto le scatole razza di scemo".split())

# Passiamo ora un parametro diverso dallo spazio. Non essendo qui presenti tre Q,

# Il valore di ritorno del metodo split sarà una lista contenente un solo elemento,

# E tale elemento non è altri che la nostra intera stringa "Hai rotto le scatole razza di scemo"

print("Hai rotto le scatole razza di scemo".split("QQQ"))

# Utilizziamo ora un separatore diverso dallo spazio. Il valore di ritorno

# sarà una lista formata da 7 elementi di tipo stringa

print("HaiQQQrottoQQQleQQQscatoleQQQrazzaQQQdiQQQscemo".split("QQQ"))

# Inseriamo ora anche il secondo argomento. Supponiamo di voler passare

# l'argomento 4. L'output sarà il seguente: ['Hai', 'rotto', 'le', 'scatole', 'razzaQQQdiQQQscemo']

print("HaiQQQrottoQQQleQQQscatoleQQQrazzaQQQdiQQQscemo".split("QQQ",4))

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**SORTED() E SORT()**

**SORTED()**

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Vediamo per prima cosa il caso in cui tale funzione restituisca errore, ossia nel caso in cui l’iterabile di partenza, passato come input, sia composto da elementi di tipo diverso.

Codice Python:

boh = ["ciao12%%&/$","cara"]

print(boh)

print()

boh\_sorted = sorted(boh)

print(boh\_sorted)

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

In questo caso il reverse è andato a buon fine, perché gli elementi dell’iterabile di partenza, anche se ciascuno di essi è composto da caretteri di diverso tipo, sono comunque stringhe. Il problema sorge se gli elementi che compongono l’iterabile di partenza non sono tutti dello stesso tipo:

Codice Pyrhon:

boh = ["ciao12%%&/$",5,"cara",True,"ok"]

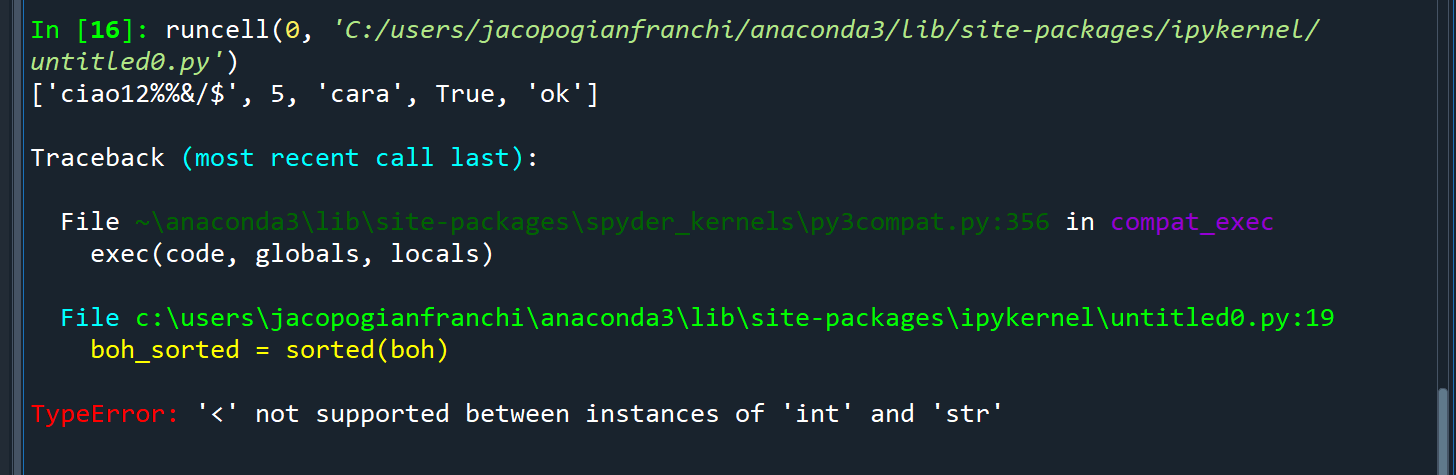
print(boh)

print()

boh\_sorted = sorted(boh)

print(boh\_sorted)

Console:



CVD.

Proviamo invece con tipi numerici. Se abbiamo tutti numeri, la funzione reversed andrà a buon fine, anche se i numeri sono di diverso tipo:

Codice Python:

boh = [12,4,5.5,8]

print(boh)

print()

boh\_sorted = sorted(boh)

print(boh\_sorted)

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

E se provassimo ad inserire numeri e booleans? Ovviamente il sort andrà a buon fine, perché il boolean sarà convertito in 0 oppure 1:

Codice Python:

boh = [12,4,5.5,8,True]

print(boh)

print()

boh\_sorted = sorted(boh)

print(boh\_sorted)

print()

boh = [12,4,5.5,8,True]

print(boh)

print()

boh\_sorted = sorted(boh,reverse=True)

print(boh\_sorted)

Console:

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Proviamo adesso un caso con tutte stringhe. Ovviamente, il sort avverrà in base ai codici ASCII relativi ai caratteri che costituiscono la singola stringa. Ovviamente, una stringa può essere formata da lettere, numeri e simboli. L’importante è che ogni eleemnto sia di tipo stringa, a prescindere dal tipo di carattere che compone la stringa.

Codice Python:

# METODO lista.sorted(): Prende in input una lista e il suo  valore di ritorno

# E' una nuova lista avente gli elementi ordinati in senso crescente in base

# Ai codici ascii corrispondenti alle lettere

cancelleria = ["Matita","Penna","Quaderno","taglierino","Pentium4","Stampante","mouse","cellulare","Tablet","3Pile+"]

print(cancelleria)

print()

cancelleria\_sorted = sorted(cancelleria)

print(cancelleria\_sorted)

Console:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Per effettuare questa conversione, Python ha per prima cosa convertito le nostre stringhe in codice ASCII. Poiché gli ASCII sono int, python ha poi ordinato numeri int dal più piccolo al più grande. Dopodiché ha riconvertito questi numeri in simboli. Vediamo un po dal punto di vista tecnico come ha ragionato python.

L’esempio è abbastanza complicato da fare, comunque si prende in considerazione il primo carattere di ciascun elemento, si converte in ASCII e, alla fine, si riordinano in ordine crescente. Se ci sono due elementi con stessa iniziale, si prende in considerazione il secondo elemento, e così via.

**SORT()**

Questo metodo è identico al metodo sorted(), ma modifica la lista di partenza

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Codice Python:

cancelleria = ["Matita","Penna","Quaderno","taglierino","Pentium4","Stampante","mouse","cellulare","Tablet","3Pile+"]

print(cancelleria)

print()

cancelleria.sort()

print(cancelleria)

Console:

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

**REPLACE()**

Il metodo replace crea una nuova stringa, sostituendo una porzione di testo della stringa originale.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Vediamo un banale esempio.

Codice Python:

frase = "Ho deciso di comprarmi un bufalo ed addomesticarlo"

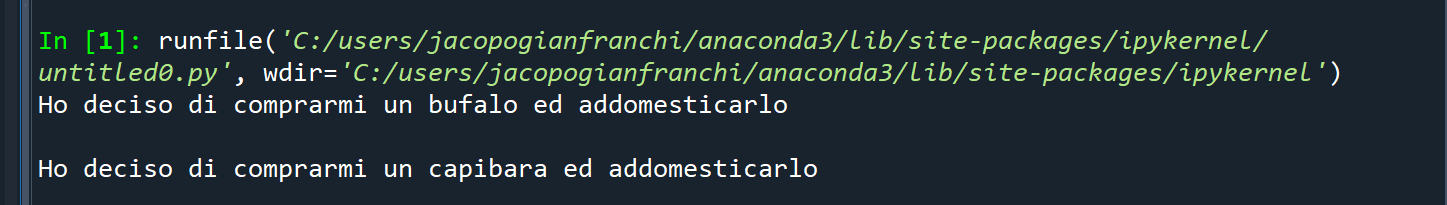
print(frase)

print()

frase\_modificata = frase.replace("bufalo","capibara")

print(frase\_modificata)

Console:



Il metodo replace è molto utile se utilizzato con il metodo join. Supponiamo di voler creare una nuova lista facendo si che ogni elemento di essa sia una parola di una frase.

Facciamo questo esercizio evitando di utilizzare il metodo replace.

Codice Python:

frase = "Che bello!!! Ieri ho comprato una mela, una pera, un kiwi. C'era lo sconto del 70%"

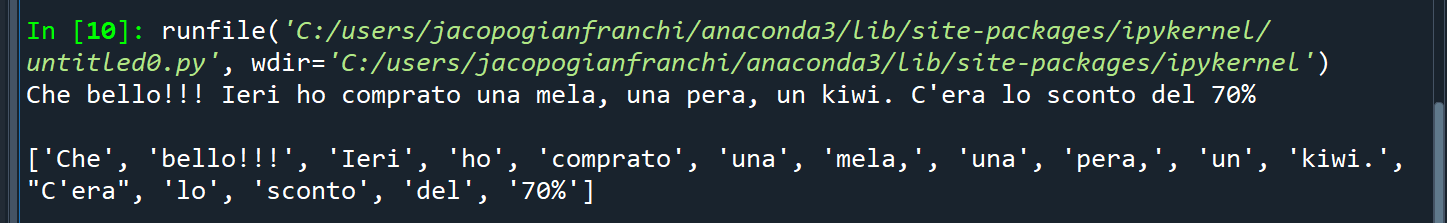
print(frase)

print()

frase\_lista = frase.split(" ")

print(frase\_lista)

Console:



L’output è parecchio bruttino, perché i caratteri speciali restano attaccati alla parola di sinistra.

Come facciamo ad eliminarli? Utilizziamo il metodo replace()!!

Codice Python:

frase = "Che bello!!! Ieri ho comprato una mela, una pera, un kiwi. C'era lo sconto del 70%"

print(frase)

print()

frase\_lista = frase.replace("!"," ")

frase\_lista = frase\_lista.replace(","," ")

fraìse\_lista = frase\_lista.replace("%", " ")

frase\_lista= frase\_lista.split()

print(frase\_lista)

Console:

