LINGUAGGIO PYTHON
FUNZIONI E PROCEDURE

SISTEMI INFORMATIVI E DBMS

CORSO DI LAUREA MAGISTRALE IN
MANAGEMENT E MONITORAGGIO DEL TURISMO SOSTENIBILE

PROF. ANDREA PINNA

AA 2019/2020

MANIPOLAZIONE STRINGHE

Introduciamo alcuni **metodi** delle stringhe. I metodi si applicano al nome della variabile.

```
.split(str="", num=string.count(str))
.count(str, beg = 0,end = len(string))
.find(str, beg = 0 end = len(string))
.lower()
.upper()
.lstrip()
```

Per conoscere il numero di caratteri si può usare la funzione len

```
len(str)
```

https://www.tutorialspoint.com/python3/python_strings.htm

MANIPOLAZIONE STRINGHE

Split è un metodo che divide una stringa in più "parole". Se utilizzato senza argomenti, il metodo divide la stringa in base alla posizione degli spazi.

```
>>> a="ciao, sono Andrea"
>>> a.split()      #Lista di parole
['ciao,', 'sono', 'Andrea']
```

È possibile specificare il carattere (o la stringa) che fa da separatore. Ad esempio, se il separatore è la virgola:

```
>>> a.split(",")  #Lista di parole separate da ","
['ciao', ' sono Andrea']
```

MANIPOLAZIONE STRINGHE

I metodi `count` e `find` di una stringa prendono ingresso un'altra stringa e valutano se questa è una sua sottostringa.

`count` restituisce il numero di corrispondenze.

```
>>> a.count("no") #conta le occorrenze della stringa "no"
1
```

`find` restituisce la posizione della prima corrispondenza (partendo da zero)

```
>>> a.find("And")  #posizione della sottostringa "And"
11
```

MANIPOLAZIONE STRINGHE

I metodi `upper` e `lower` consentono di portare tutte le lettere allo stesso "case" (o tutte maiuscole o tutte minuscole). È utile quando voglio esaminare il contenuto della stringa a prescindere dal case.

Esempio:

```
>>> a.upper()
'CIAO, SONO ANDREA'

>>> a.lower()
'ciao, sono andrea'
```

MANIPOLAZIONE STRINGHE

Il metodo `lstrip` non ha argomenti e restituisce la stringa rimuovendo tutti i caratteri di spazio che precedono il primo carattere diverso dallo spazio.

Esempio:

```
>>> unaStringa = '   ciao'
>>> b = unaStringa.lstrip()
>>> print(b)
ciao
```

FUNZIONI E PROCEDURE

Nei linguaggi di programmazione la funzione e la procedura sono entrambe una sequenza di istruzioni ideata per eseguire una specifica elaborazione per la quale si definisce un nome.

La sequenza di istruzioni può essere richiamata più volte nel programma (usando il nome) e permette di evitare la ripetizione dello stesso gruppo di istruzioni. Consente inoltre di migliorare la leggibilità del codice.

FUNZIONI E PROCEDURE

Tutte le funzioni e procedure che abbiamo visto fino ad ora sono predefinite in python. In genere queste funzioni si chiamano funzoni di built-in.

La documentazione Python fornisce la lista e la descrizione delle funzioni di built-in.

https://docs.python.org/3/library/functions.html

Altre funzioni di built-in sono contenute nelle librerie (o moduli).

FUNZIONI E PROCEDURE

Una funzione è una sequenza di istruzioni che può prendere in ingresso una lista di argomenti e produce un risultato (return). Il risultato può essere assegnato ad una variabile e la funzione può far parte di una espressione.

Esempio la funzione di built-in `max`:

```
mioRisultato = max(1, 2, 3)
```

FUNZIONI E PROCEDURE

Una procedura è una sequenza di istruzioni che può prendere in ingresso una lista di argomenti e NON produce un risultato da ritornare.

Esempio la funzione di built-in print:

```
risultato = print('ciao')
```

L'istruzione valida ma otteniamo che la variabile `risultato` non ha valore e risulta di tipo `'NoneType'`

DEFINIZIONE

Con Python funzioni e procedure si creano allo stesso modo, tramite la dichiarazione. Vogliamo ora dichiarare una nuova funzione. Sintassi:

```
def nomeFunzione ( Lista di parametri):
    istruzioni #blocco di codice tabulato
```

Il nome della funzione rispetta le stesse regole dei nomi delle variabili (inizia con una lettera e sono vietate le parole chiave).
Il corpo della funzione è un blocco di istruzioni tabulato.

DEFINIZIONE

La lista dei parametri è una lista di nomi di variabile separati da una virgola. Un parametro è un nome che il programmatore definisce per raccogliere un argomento "passato" alla funzione, in base all'ordine di inserimento.

Esempio:

```
def stampaSomma(valore1, valore2):
    print(valore1+valore2)

stampaSomma(2,1) #valore1 prende 2, valore2 prende 1
stampaSomma(2*2,13/2)
mioDato = 12
stampaSomma(mioDato, 5)
```

DEFINIZIONE DI FUNZIONI

Una funzione restituisce un valore solo se l'ultima istruzione da essa eseguita è `return`.

`return` è una parola chiave. Indica all'interprete che la funzione deve assumere un valore pari a quello dell'espressione a destra di `return`. Quando l'interprete incontra l'istruzione `return` interrompe sempre l'esecuzione della funzione (anche se dentro una struttura di controllo).

```
def nomeFunzione(parametri):
    istruzioni
    return espressione
```

DEFINIZIONE DI FUNZIONI

Esempio: creo la funzione inverti che restituisce il valore passato come argomento cambiato di segno.

```
def inverti(mioNum):
    return -mioNum

Esempio d'uso:

ris = inverti(2) #ris assume valore -2
ris = inverti(ris) #ora ris è 2
```

DEFINIZIONE DI FUNZIONI

Le espressioni possono essere lunghe a piacere, purché scritte rispettando le regole del linguaggio. Lo stile di programmazione è una scelta del programmatore. Da esso dipende la leggibilità del codice. (Zen Python)
Esempio di poca leggibilità:

```
def miaFunzione(var1, var2):
    return funz(funz2("un " + var2), var1) + 123
```

In questo esempio vengono richiamate due funzioni dentro la funzione. Supponiamo che le funzioni `funz` e `funz2` siano già definite nel programma prima di `miaFunzione`.

DEFINIZIONE DI FUNZIONI

Questa funzione ha lo stesso comportamento ma risulta più leggibile perché fa uso di più variabili:

```
def miaFunzione(var1, var2):
    stringa = "un " + var2
    risParziale1 = funz2(stringa)
    risParziale2 = funz(risParziale1 , var1)
    risultato = risParziale2 + 123
    return risultato
```

DEFINIZIONE DI FUNZIONI

Valori di default.
Posso dichiarare una funzione aggiungendo un valore pre-impostato (o valore di default) ai vari parametri.
Esempio:

```
def stampaCoordinate(x=0,y=0,z=0):
    print("x,y,z = ",x,y,z)
```

In questo modo posso richiamare la funzione passando soltanto un sottoinsieme di argomenti o nessuno.

DEFINIZIONE DI FUNZIONI

Esempio:

```
stampaCoordinate()
x,y,z = 0 0 0
stampaCoordinate(11) #Il parametro x prende 11
x,y,z = 11 0 0
```

Se salto un parametro o se cambio l'ordine, per evitare ambiguità è necessario specificare a quale parametro assegnare il valore. Sintassi:

```
stampaCoordinate(z=1,y=33) #x resta 0
x,y,z = 0 33 1
```

SPAZIO DEI NOMI

I nomi di variabile definiti all'interno di una funzione sono detti variabili locali e non esistono al di fuori della funzione.

Viceversa, i nomi di variabili definiti al di fuori di una funzione sono detti variabili globali ed esistono anche dentro la funzione ma il loro valore non può essere modificato.

SPAZIO DEI NOMI

Esempio: creo una variabile globale `a` con valore 10. Dentro la funzione creo la variabile locale `b` con valore 15 .

```
a = 10
def miaProcedura():
    b = 15  #Variabile LOCALE
    print(a+b)
miaProcedura()
print(b)
```

Il programma stamperà a video il valore 25 ma poi darà errore perché non trova la variabile b.

SPAZIO DEI NOMI

Esempio: provo a modificare la variabile globale all'interno della funzione.

```
a = 10
def miaProcedura():
    a = 11
    print(a)
miaProcedura()
```

La viariable gloabale a non subirà nessun effetto e resterà del valore 10.

SPAZIO DEI NOMI: GLOBAL

Tramite la parola chiave **`global`** impongo all'interprete di elaborare come globale la variabile dentro la funzione.

```
a = 10
def miaProcedura():
    global a
    a = 11
    print(a)
miaProcedura()
print(a)
```

Ora il valore di `a` è modificato globalmente (anche al di fuori della funzione).

SPAZIO DEI NOMI: GLOBAL

Con global posso creare variabili globali all'interno delle funzioni.

```
def miaProcedura():
    global a
    a = 11

#print(a) #DAREBBE ERRORE
miaProcedura()
print(a) #NESSUN ERRORE
```

La variabile `a` inizia ad esistere globalmente soltanto dopo aver chiamato la funzione `miaProcedura`.

SPAZIO DEI NOMI: DEL

La parola chiave `del` rimuove permette di rimuovere una variabile o una funzione dallo spazio dei nomi.
Esempio:

```
>>> x = 35
>>> def stampaX():
            print(x)
>>> stampaX()
35
>>> del x #Elimina il nome x
>>> stampaX()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in func
NameError: name 'x' is not defined
```

L'interprete non trova più la variabile x tra i nomi conosciuti.

ISTRUZIONE PASS

La parola chiave `pass` indica all'interprete di non fare nulla e passare all'istruzione successiva.

Esempio 1:

```
for i in range(10):
    if(i==3):
        pass
    else:
        print(i)
```

Esempio 2:

```
try:
    a = int(valore)
except:
    pass
```

ISTRUZIONE BREAK

La parola chiave `break` si usa dentro i cicli e serve a chiedere all'interprete di interrompere forzatamente il ciclo in corso.

Esempio 1:

```
for i in range(10):
    if(i==3):
        break
    else:
        print(i)
```

È utile quando non è possibile stabilire in anticipo quando arrestare il ciclo.

```
a = 0
while True:
    a = int(input("scrivi un numero positivo:"))
    if a > 0:
        break
```