Cheatography

Imparare Python Cheat Sheet

by GtM via cheatography.com/215027/cs/46822/

Variabili

Nomenclatura	snake_case che inizia per _ o lettera
Assegnazione singola	`nome_v ari abile = valore`
Assegnazione multipla	`var1, var2, var3 = val1, val2, val3` `var1 = var2 = var3 = valore`

Sintassi di base

Commento

```
# Il tuo commento qui
```

Stampare a schermo

```
print()
```

Indentazione

Le righe aventi stessa indentazione appartengono allo stesso blocco di codice.

Tipi di dati

Numerici	int, float, complex, bool	
Stringhe	str (immutabile)	
Sequenze	list (ordinata, mutabile, permette duplicati) tuple (ordinata, immutabile, permette duplicati) range	[] ()
Insiemi	set (non ordinato, mutabile, non permette duplicati) frozenset (non ordinato, immutabile, non permette duplicati)	{}
Dizionari	dict (chiave-valore)	
Binari	bytes, bytearray, memoryview	

Valore di verità

True

Un oggetto è considerato vero a meno che la sua classe non definisca un metodo *__bool__()* che restituisce False o un metodo *__len__()* che restituisce zero.

False

> Costanti None e False

> Zero di qualsiasi tipo numerico (0, 0.0 ecc.)

> Sequenze o collezioni vuote ("", (), [] ecc.)

Operatori di confronto

<	Strettamente minore	`1 < 2  # True`
<=	Minore o uguale	`1 <= 1  # True`
>	Strettamente maggiore	`4 > 2  # False`
>=	Maggiore o uguale	`3 >= 2  # True`
==	Uguale	`5 == 7  # False`
!=	Diverso	`1 != 2  # True`

is	Identità dell'oggetto	`a = [1,2]`
		`b = a`
		`print(a is b) # True`

is not	Identità dell'oggetto negata	`a = [1,2]`
		`b = a`
		`print(a is not b) # False`

Possono essere concatenati arbitrariamente:
X < Y <= Z equivale a X < Y AND Y <= Z

Operatori numerici

x + y	Addizione	`3 + 4 = 7`
x - y	Sottrazione	`5 - 2 = 3`
x * y	Prodotto	`7 * 1 = 7`
x / y	Divisione	`8 / 4 = 2`
x // y	Divisione arrotondata per difetto	`3 // 2 = 1` `-1 // 2 = -1`
x % y	Modulo (ritorna il resto della divisione)	`7 % 3 = 1`
-x	Negazione	`-5`
abs(x)	Valore assoluto	`abs(-5.5) = 5.5`
int(x)	Conversione in int	`int("12 ") = 12` `int(7.2) = 7`
float(x)	Conversione in float	`float(7) = 7.0`
complex(re, im)	Conversione in numero complesso	`comple x("3 + 5j")`
divmod(x, y)	Ritorna una tupla contenente quoziente intero e modulo dei parametri	`divmod(5, 2) = (2, 1)`

Operatori numerici (cont)

pow(x, y)	Potenza	`pow(2, 3) = 8`
x ** y		`2 ** 3 = 8`

Quando gli operandi sono di tipi numerici differenti, il tipo più piccolo viene convertito in quello più grande (int < float < complex).

Operatori booleani

x or y	True se almeno uno tra x e y è True, altrimenti False.	Non valuta il secondo argomento se il primo è True.
x and y	True se entrambi x e y sono True, altrimenti False.	Non valuta il secondo argomento se il primo è False.
not x	True se x è False, altrimenti False.	Ha priorità inferiore rispetto agli operatori non booleani *

* not a == b -> not (a == b)

Tipi Numerici

int	Numeri interi a precisione illimitata. Costruttore -> *int()*
float	Numeri in virgola mobile, si utilizza il punto tra parte intera e parte decimale. Costruttore -> *float()*
complex	Numeri complessi, composti da una parte reale (n.real) ed una parte immaginaria (n.imag). Costruttore -> *complex()*
bool	Valori di verità (True e False).

Stringhe

Assegnazione	`stringa = "Hello"` `stringa = 'Hello'`	

Stringa multiriga	`"""Hello,`
	`it's`
	`me"""`	

Accedere ai caratteri	`stringa[0]` `stringa[-1]`	`# primo carattere` `# ultimo carattere`
Lunghezza	`len("He llo ")`	`# 5`

Stringhe (cont)

Sottostringa	`stringa[:n]`	`# da inizio a _n`
	`stringa[n:]`	`# da °n a fine`
	`stringa[n:m]`	`# da °n a _m`
Concatenazione	`" ab" + " cd"`	`# " abc d"`
Ripetizione	`" " Hel lo"\ * 2`	`#"He llo Hel lo"`
Formattazione	`f"Ho {10} mele"`	`# "Ho 10 mele"`
Caratteri di escape	`\n`	`# new line`
	`\t`	`# tab`
	`\'`	`# '`

°x -> incluso / _x -> escluso

Metodi delle stringhe

capitalize()	`" heL Lo".c ap ita lize()`	`# " Hel lo"`
casefold()	`" HeL Lo".c as efold()`	`# " hel lo"`
center()	`" Hel lo".c en ter (7, " -")`	`# " -He llo -"`
count()	`" Hel lo".c ou nt( " l")`	`# 2`
endswith()	`"Hello".endswith("lo")`	`# True`
	`"Hello".endswith("ab")`	`# False`
expandtabs()	`" 1\t 2".e xpa ndt abs(1)`	`# "1 2"`
find()	`" Hel lo".f in d("l ")`	`# 2`
format()	`" Hel {}".f or mat ("lo ")`	`# " Hel lo"`
format_map()	come *format()*	Accetta dizionari come parametri
index()	come *find()*	ValueError se non trova la sottostringa
isalnum()	`"ABC123".isalnum()`	`# True`
	`"ab!12_".isalnum()`	`# False`
isalpha()	`"Abc".isalpha()`	`# True`
	`"123".isalpha()`	`# False`
isascii()	`"Abc123".isascii()`	`# True`
	`"Ab 23".i sa scii()`	`# False`
isidentifier()	`"hello".isidentifier()`	`# True`
	`"123ab".isidentifier()`	`# False`

Metodi delle stringhe (cont)

islower()	`"hello1".islower()`	`# True`
	`"Hello1".islower()`	`# False`
isprintable()	`" Abc 1!".i sp rin tab le( ){n l}} " A\t 1!".i sp rin ta ble()`	`# True` `# False`
isspace()	`"\t\n ".isspace()`	`# True`
	`"a  b".i ssp ace()`	`# False`
join()	`" _".j oin (["a " ,"b"])`	`# " a_b "`
ljust()	`" Abc ".lj ust (5, " -")`	`# " Abc --"`
lower()	`" HEL LO".l ower()`	`# " hel lo"`
lstrip()	`" -He llo -".l str ip( " -")`	`# " Hel lo- "`
maketrans() **translate()**	`map1 = str.maketrans({"a":1,"b":2})})` `map2 = str.maketrans("ab","12")` `map3 = str.maketrans("ab","12","c")`	
	`"abc".translate(map1)`	`# "12c"`
	`"abc".translate(map2)`	`# "12c"`
	`"abc".translate(map3)`	`# " 12"`
partition()	`" a-b -c".p ar tit ion()`	`# ("a", " -","b -c")`
replace()	`" abc ".re pla ce( " b","0 ")`	`# " a0c "`
rfind()	`" Hel lo".r fi nd( " l")`	`# 3`
rindex()	come *rfind()*	ValueError se non trova la sottostringa
rjust()	`" Abc ".rj ust (5, " -")`	`# " --A bc"`
rpartition()	`" a-b -c".r pa rti tion()`	`# ("a- b","- " ,"c")`
rsplit()	`" a-b -c".r sp lit ("-",1)`	`# ["a- b","c "]`
rstrip()	`" -He llo -".r str ip( " -")`	`# " -He llo "`
split()	`" a-b -c".s pl it( " -",1)`	`# ["a", " b-c "]`
splitlines()	`" a\n b".s pli tli nes()`	`# ["a", " b"]`
startswith()	`"abc".startswith("a")`	`# True`
	`"abc".startswith("c")`	`# False`
strip()	`" -He llo -".s tri p("- ")`	`# " Hel lo"`
swapcase()	`" XxX ".sw apc ase()`	`# " xXx "`

Metodi delle stringhe (cont)

title()	`" HELLO world".t itle()`	`# " Hello World"`
upper()	`" hel lo".u pper()`	`# " HEL LO"`
zfill()	`"1".zfill(3)`	`# "001"`
	`"-1".zfill(3)`	`# " -00 1"`
isdigit() **isnumeric()** **isdecimal()**	`# Numeri standard "123"`	
	`isdigit()`	`# True`
	`isnumeric()`	`# True`
	`isdecimal()`	`# True`
	`# Esponenti "¹²³"`	
	`isdigit()`	`# True`
	`isnumeric()`	`# True`
	`isdecimal()`	`# False`
	`# Ideogrammi numerici`	
	`isdigit()`	`# False`
	`isnumeric()`	`# True`
	`isdecimal()`	`# False`
	`# Numeri decimali "12.34"`	
	`isdigit()`	`# False`
	`isnumeric()`	`# False`
	`isdecimal()`	`# False`
	`# Numeri negativi "-123"`	
	`isdigit()`	`# False`
	`isnumeric()`	`# False`
	`isdecimal()`	`# False`
	`# Stringa vuota ""`	
	`isdigit()`	`# False`
	`isnumeric()`	`# False`
	`isdecimal()`	`# False`

* più aggressivo rispetto a lower()

Imparare Python Cheat Sheet

by GtM via cheatography.com/215027/cs/46822/

Liste

Assegnazione	`lista = ["a", 2, True, [1, 2], ...]`	
Accedere agli elementi	`lista[0]`	`# primo elemento`
	`lista[-1]`	`# ultimo elemento`
Lunghezza	`len(["a " ,"b"])`	`# 2`
Concatenazione	`[1, 2] + [3, 4]`	`# [1, 2, 3, 4]`
Ripetizione	`[1, 2] * 2`	`# [1, 2, 1, 2]`
Sottolista	`lista[:n]`	`# da inizio a _n`
	`lista[n:]`	`# da °n a fine`
	`lista[n:m]`	`# da °n a _m`

°x -> incluso / _x -> escluso

Metodi delle liste

append()	`[1, 2].app end(3)`	`# [1, 2, 3]`
clear()	`[1, 2].clear()`	`# []`
copy()	`l1 = [1, 2]`	
	`l2 = l1.copy()`	`# l1 e l2 sono due liste distinte`
	`l3 = l1`	`# l1 e l3 puntano alla stessa lista`
count()	`[1, 2, 1].cou nt(1)`	`# 2`
extend()	`[1, 2].ext end (["a ", " b"])`	`# [1, 2, " a", " b"]`
index()	`[1, 2, 1].ind ex(1)`	`# 0`
insert()	`[1, 2].ins ert (1,3)`	`# [1, 3, 2]`
pop()	`[1, 2].pop(0)`	`# [2] , ritorna 1`
remove()	`[1, 2, 1].rem ove(1)`	`# [2, 1]`
reverse()	`[1, 2, 3].rev erse()`	`# [3, 2, 1]`
sort()	`[2, 1, 3].sort()`	`# [1, 2, 3]`
	`[2, 1, 3].sor t(r eve rse =True)`	`# [3, 2, 1]`

Tuple

Assegnazione	`tupla = ("a", 2, True, ...)`	
	`stringa = (1, )`	`# singolo elemento`
Accedere agli elementi	`tupla[0]`	`# primo elemento`
	`tupla[-1]`	`# ultimo elemento`
Lunghezza	`len((1,2))`	`# 2`

Tuple (cont)

Sottotupla	`tupla[:n]`	`# da inizio a _n`
	`tupla[n:]`	`# da °n a fine`
	`tupla[n:m]`	`# da °n a _m`
Concatenazione	`(1, 2) + (3, 4)`	`# (1, 2, 3, 4)`
Ripetizione	`(1, 2) * 2`	`# (1, 2, 1, 2)`

°x -> incluso / _x -> escluso

Metodi delle tuple

count()	`(1, 2, 1).cou nt(1)`	`# 2`
index()	`(1, 2, 3).ind ex(3)`	`# 2`

Range

range(start, stop, step)

start	Opzionale	Numero iniziale (compreso, default = 0)
stop	Obblig-atorio	Numero finale (escluso)
step	Opzionale	Passo dell'incr-emento (default = 1)

IF

Sintassi

```
if condiz ione1:
    blocco_1
elif condiz ione2:
    blocco_2
...
else:
    blocco _else
```

Singola istruzione

```
if condizione: istruzione
```

Operatore ternario

```
se_true if condizione else se_false
```

pass*

```
if condizione:
    pass
```

* i blocchi non possono essere vuoti

MATCH

Sintassi

```
match espres sione:
    case valore1:
        blocco_1
    case valore2:
        blocco_2
    ...
    case _:
        blocco _de fault
```

Combinazione di valori

```
case valore1 | valore2 | ... :
```

Case ... if

```
match len(stringa):
    case 4 if stringa[0] == "a":
        blocco_1
    ...
```

Case e sequenze

```
match sequenza:
    case [x, y]:
        print( f"Lista con due elementi: {x}, {y}")
    case [x, y, z]:
        print( f"Lista con tre elementi: {x}, {y}, {z}")
    case _:
        print( " Formato sconos ciu to")
```

Case e dizionari

```
match dizionario:
    case {"no me": nome, " eta ": eta}:
        print( f"Nome: {nome}, Età: {eta}")
    case {"no me": nome}:
        print( f"Nome: {nome}")
    case _:
        print( " Formato sconos ciu to")
```

WHILE

Sintassi

```
while condizione:
    blocco _codice
else:
    blocco _else
```

break

Interrompe il loop

WHILE (cont)

continue	Salta l'iterazione corrente del loop

FOR

Sintassi	`for elemento in sequenza:`
	`  blocco _codice`
	`else:`
	`  blocco _else`

For each	`for carattere in stringa:`
	`  print(carattere)`
	`else:`
	`  print( " Ciclo termin ato ")`

Range	`for i in range(5):`
	`  print(stringa[i])`
	`else:`
	`  print( " Ciclo termin ato ")`

break	Interrompe il loop
continue	Salta l'iterazione corrente del loop
pass*	`for elemento in sequenza:` `  pass`

* i blocchi non possono essere vuoti

Funzioni

Sintassi	`def`

Funzioni

Sintassi	`def nome_f unzione(parametri):` `  blocco _codice`	return opzionale

Args (n parametri)	`def contaS tri nghe(*stringhe):`
	`  return len(stringhe)`
	`contaStringhe("a")`
	`contaStringhe("a", " b")`
		
		`# 1`
		`# 2`

Funzioni (cont)

Keyword

```
def ordina Ele men ti( primo, secondo):
    print( f"Or dine: {primo} e {secondo})
ordinaElementi(1, 2)                                    # " Ordine: 1 e 2"
ordinaElementi(2, 1)                                    # " Ordine: 2 e 1"
ordinaElementi(secondo = 2, primo = 1)                  # " Ordine: 1 e 2"
```

Kwargs (n chiave-valore)

```
def concatenaCV(**kwargs):
    concat Chiavi = ""
    for arg in kwargs:
        concat Chiavi += arg
    concat Valori = ""
    for val in kwargs.values():
        concat Valori += val
    print( f"{c onc atC hiavi} e {concatValori}")
concatenaCV(a = 1, b = 2, c = 3)                        # "abc e 123"
```

Default

```
def stampa Naz ion e(n azione = "Italia"):
    print(nazione)
stampaNazione()                                         # "Italia"
stampaNazione("Germania")                               # " Ger man ia"
```

pass*

```
def funzione():
    pass
```

Solo parametri posizionali

```
def funzio ne( par ametri, /):
    blocco _codice
```

Solo parametri con chiavi

```
def funzione(*, parametri):
    blocco _codice
```

Parametri misti

```
def funzio ne( pos1, pos2, /, *, chiave1, chiave2):
    blocco _codice
```

* i blocchi non possono essere vuoti