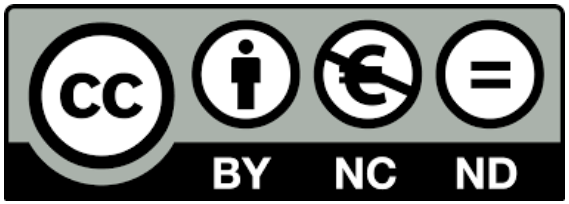
Il linguaggio Python (ver. 3)

ITS - Umbria

A.S. 2022-23

MATERIA: Fondamenti di informatica

Docente: prof. Paolo Bernardi



Il linguaggio Python (ver. 3)

Cos'è Python?

Python è un linguaggio di **programmazione ad alto livello**, rilasciato pubblicamente per la prima volta nel **1991** dal suo creatore **Guido van Rossum**, programmatore olandese attualmente operativo in **Dropbox**.

Deriva il suo nome dalla commedia Monty Python's Flying Circus dei celebri Monty Python, in onda sulla BBC nel corso degli anni 70.

Attualmente, lo sviluppo di Python (grazie e soprattutto all'enorme e dinamica comunità internazionale di sviluppatori) viene gestito dall'organizzazione no-profit **Python Software Foundation**.

Il linguaggio Python (ver. 3)

Cos'è Python?

Python supporta diversi paradigmi di programmazione, come quello object-oriented (con supporto all'ereditarietà multipla), quello imperativo e quello funzionale, ed offre una tipizzazione dinamica forte.

È fornito di una libreria built-in estremamente ricca, che unitamente alla gestione automatica della memoria e a robusti costrutti per la gestione delle eccezioni fa di Python uno dei linguaggi più ricchi e comodi da usare.

Comodo, ma anche semplice da usare e imparare. Python, nelle intenzioni di Guido van Rossum, è nato per essere un linguaggio immediatamente intuibile. La sua sintassi è pulita e snella così come i suoi costrutti, decisamente chiari e non ambigui. I blocchi logici vengono costruiti semplicemente allineando le righe allo stesso modo, incrementando la leggibilità e l'uniformità del codice anche se vi lavorano diversi autori.

Cos'è Python?

Python è un linguaggio pseudocompilato: un interprete si occupa di analizzare il codice sorgente (semplici file testuali con estensione .py) e, se sintatticamente corretto, di eseguirlo. In Python, non esiste una fase di compilazione separata (come avviene in C, per esempio) che generi un file eseguibile partendo dal sorgente.

L'essere pseudointerpretato rende Python un linguaggio portabile. Una volta scritto un sorgente, esso può essere interpretato ed eseguito sulla gran parte delle piattaforme attualmente utilizzate, siano esse di casa Apple (Mac) che PC (Microsoft Windows e GNU/Linux). Semplicemente, basta la presenza della versione corretta dell'interprete.

Cos'è Python?

Infine, Python è free software: non solo il download dell'interprete per la propria piattaforma, così come l'uso di Python nelle proprie applicazioni, è completamente gratuito; ma oltre a questo Python può essere liberamente modificato e così ridistribuito, secondo le regole di una licenza pienamente open-source.

Queste caratteristiche hanno fatto di Python il protagonista di un enorme diffusione in tutto il mondo, e anche in Italia, negli ultimi anni. Questo perché garantisce lo sviluppo rapido, e per certi versi divertente, di applicazioni di qualsiasi complessità in tutti i contesti: dal desktop al web, passando dallo sviluppo di videogiochi e dallo scripting di sistema.

Quali strumenti?

Per poter eseguire codice Python sui computer dovremo inizialmente scaricare e installare l'interprete: anche se ci sono due versioni di Python, la 2.x e la 3.x, se non si hanno esigenze particolari, come ad esempio la compatibilità con alcune librerie non aggiornate alla nuova versione, sarà opportuno scegliere la versione 3.x del linguaggio.

Se si sta usando una distribuzione Linux o Mac OSX (uguale o successiva alla 10.3 - Panther), prima di cercare di installare Python è opportuno controllare il sistema, perché molto probabilmente è già installato.

Quali strumenti?

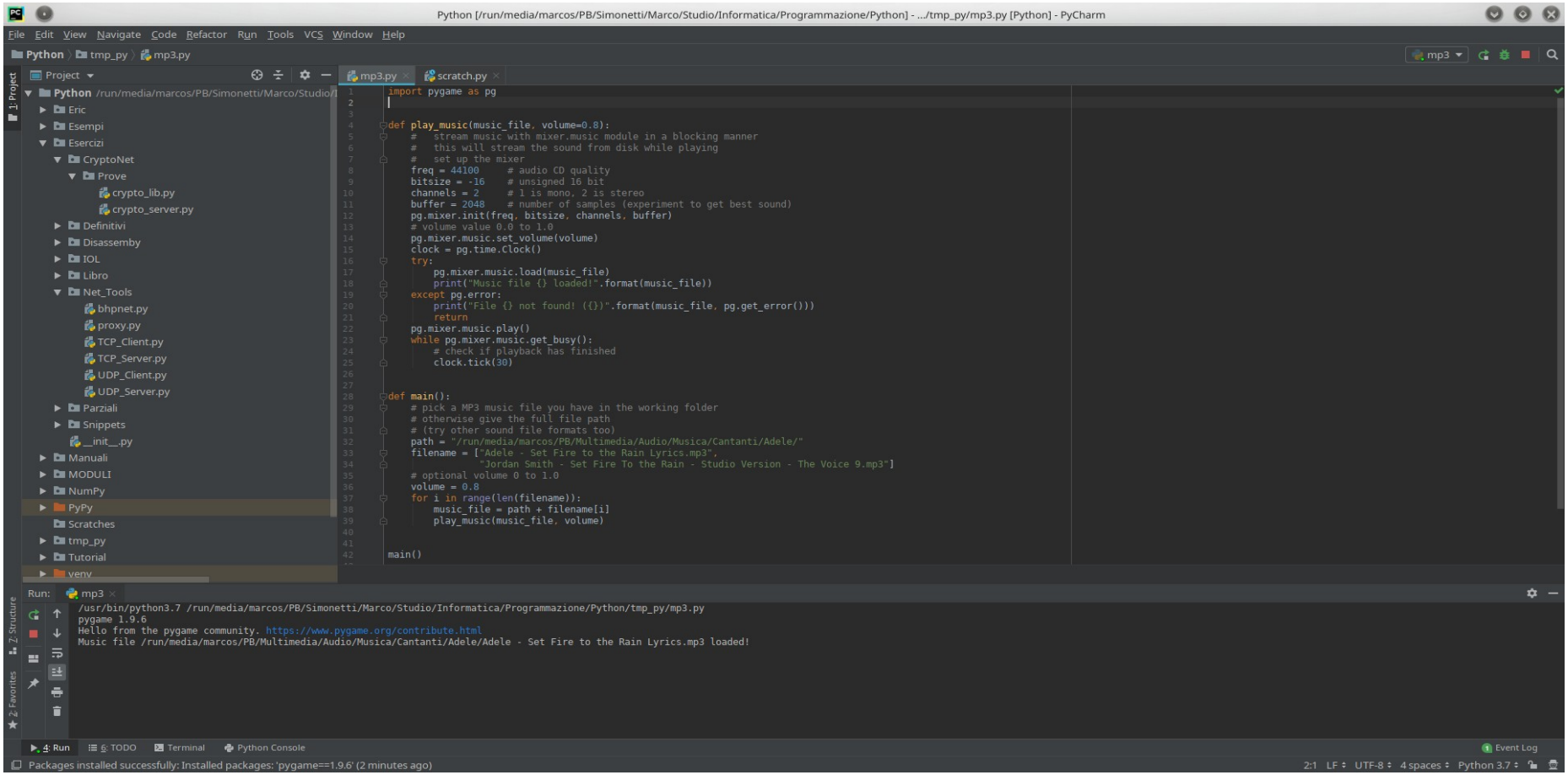
Una volta installato, è possibile usare Python direttamente in **modalità interattiva**, che permette di testare brevi frammenti di codice. C'è anche un ambiente di sviluppo in **bundle** chiamato **IDLE**.

Inoltre, è possibile elaborare script Python con un qualsiasi **programma di editing di testi** (Word, Writer, Kate, NotePad, Vim, Emacs, Sublime, Atom, ecc..) e poi eseguirlo da linea di comando.

Per i nostri scopi, utilizzeremo un **ambiente di sviluppo integrato** (IDE), come **PyCharm**, che supporta lo sviluppatore segnalando **errori di sintassi del codice** direttamente in fase di scrittura, oltre ad avere tutta una serie di strumenti e funzionalità di supporto alla **fase di sviluppo e debugging**.

Il linguaggio Python (ver. 3)

Quali strumenti?



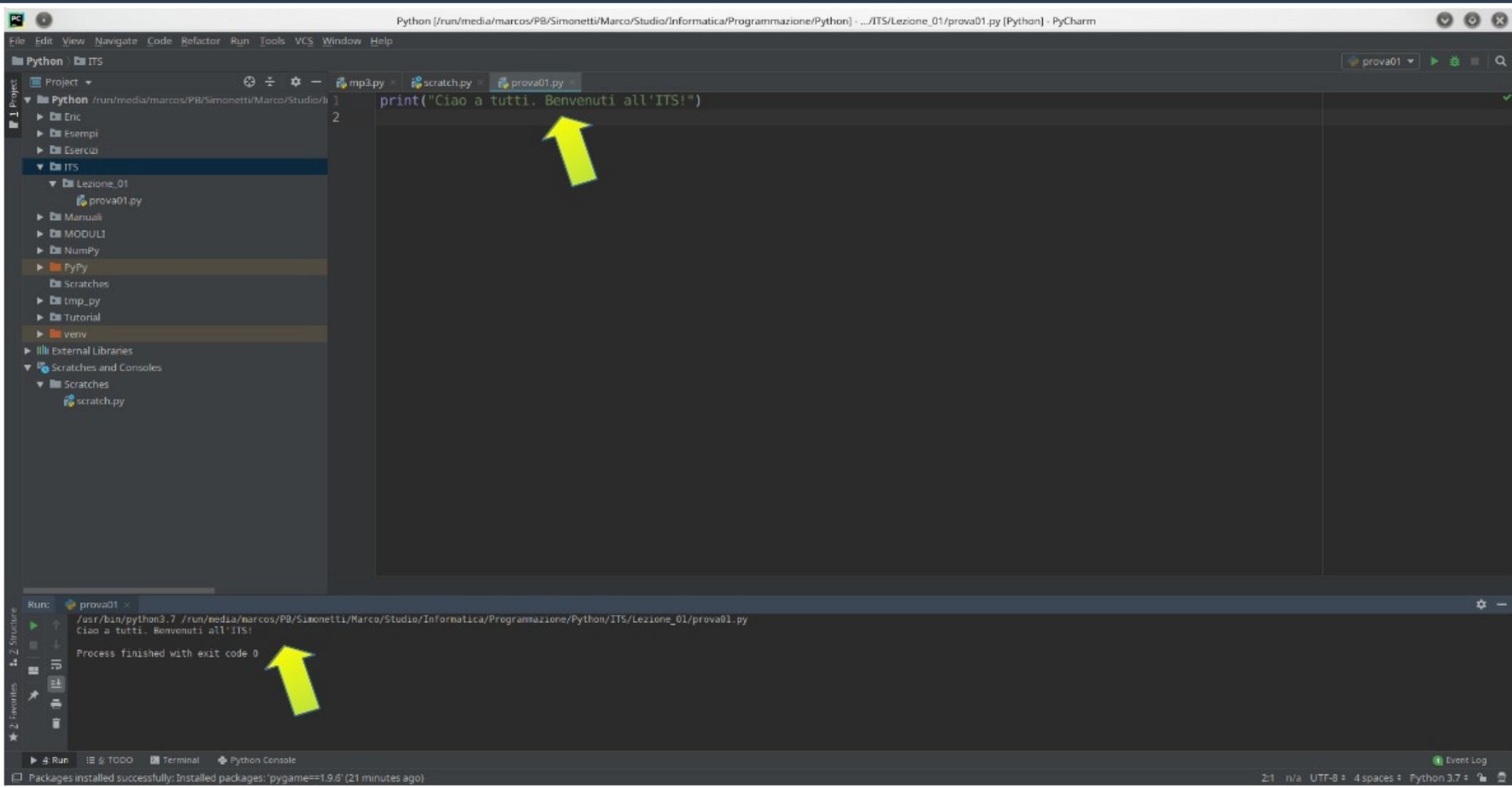
Primo programma

All'interno dell'ambiente PyCharm, creiamo una cartella chiamata ITS e al suo interno una cartella Lezione01.

In essa, scriviamo il nostro primo file Python, chiamato prova01.py, e successivamente eseguiamolo.

```
1  print("Ciao a tutti. Benvenuti all'ITS!")
2
```

Il linguaggio Python (ver. 3)



Indentazione del codice e commenti

Dove in altri linguaggi di programmazione il rientro (indentazione) del codice serve unicamente a migliorarne la leggibilità, in Python il rientro è molto importante.

Python usa il rientro per indicare un blocco di codice.

```
1 for i in range(20):
2     print("Valore di i:",i)
```

Come in altri linguaggi, è possibile "commentare" il codice, inserendo spiegazioni, informazioni accessorie e altro, direttamente tra le linee del programma.

Python usa il carattere # per commentare il codice su singola riga.

```
1 # Questo è un commento
2 print("Oggi è giovedì")
```

Documentazione Interattiva

Per ottenere una lista dei metodidi un oggetto, senza descrizione, si può usare la funzione dir.

Per ottenere la lista dei metodi delle stringhe, invochiamo `dir` su una stringa:

```
dir("biotechnology rules!")
["capitalize", "center", "count", "decode", "encode",
"endswith", "expandtabs", "find",
...
"rstrip", "split", "splitlines", "startswith", "strip",
"swapcase", "title", "translate", "upper", "zfill"]
```

(I metodi che cominciano con `"__"` non ci interessano.) Otteniamo lo stesso risultato con **qualunque** stringa:

```
dir("bioinformatics is a-okay!")
```

Documentazione Interattiva

Per accedere al manuale di un tipo/oggetto/variabile, si usi la funzione help.

Il manuale descrive i metodi supportati dal tipo dell'oggetto/variabile.

Per leggere il manuale del metodo `split` delle stringhe, scriviamo:

```
help("".split)
Help on built-in function split:

split(...)
    S.split([sep [,maxsplit]]) -> list of strings

    Return a list of the words in the string S, using sep as the
    delimiter string.  If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator and empty strings are removed
    from the result.
```

Stessa cosa con:

```
help("123".split)
```

Variabili: introduzione

Una variabile è un'etichetta per una posizione in memoria. Può essere usato per contenere un valore. Nei linguaggi tipizzati staticamente, le variabili hanno tipi predeterminati e una variabile può essere utilizzata solo per contenere valori di quel tipo. In Python, che è un linguaggio tipizzato dinamicamente, possiamo riutilizzare la stessa variabile per memorizzare valori di qualsiasi tipo.

La loro durata di vita dipende dalla posizione nel programma e dalle relative referenze attive sull'oggetto.

```
>>> a = 2
>>> a = 3.2
>>> a = "cambio valore"
>>> a = [1, 2, 3, 4]
>>> a = {"Carlo": "333", "Giovanni": "444}
```

Variabili: creazione di variabili

A differenza di altri linguaggi di programmazione, Python non ha alcun comando per dichiarare una variabile. Una variabile viene creata nel momento stesso in cui si assegna un valore ad essa. L'istruzione di assegnazione crea nuove variabili e assegna loro un valore:

```
>>> messaggio = "Come va?"
>>> n = 17
>>> pi = 3.14159
>>> a, b = 15, 34.5  # assegnazione multipla
```

Non è necessario dichiarare un tipo di dato da assegnare, poiché Python lo riconosce dall'espressione a destra dell'operatore di uguaglianza. Per verificare il tipo di dato assegnato alle tre variabili di cui sopra è sufficiente la funzione *type(nome_variabile)*.

```
>>> type(message)
<type 'string'>
>>> type(n)
<type 'int'>
>>> type(pi)
<type 'float'>
```

Variabili: cancellazione di variabili

Per cancellare (dereferenziare ed eliminare) una variabile si può usare l'istruzione **del** seguita dal nome, o dai nomi, della variabile.

Es:

```
>>> del messaggio

>>> del a, b   # cancellazione multipla
```

Variabili: Nomi delle variabili e parole riservate

I programmatori generalmente scelgono dei nomi significativi per le loro variabili, documentando così a che cosa servono.

I nomi delle variabili possono essere lunghi quanto si desidera e possono contenere sia lettere che numeri, ma devono sempre iniziare con una lettera.

È legale usare sia lettere maiuscole che minuscole, ma per l'interprete esse saranno variabili diverse, così che Numero, NUmEro e numero sono a tutti gli effetti variabili diverse (linguaggio Case Sensitive).

Il carattere di sottolineatura (_) può far parte di un nome ed è spesso usato in nomi di variabile composti da più parole (per esempio: il_mio_nome e prezzo_del_the).

In alternativa le parole possono essere composte usando l'iniziale maiuscola per ciascuna di esse, con il resto dei caratteri lasciati in minuscolo come in IlMioNome e PrezzoDelThe.

Assegnando un nome illegale alla variabile, si otterrà un messaggio d'errore di sintassi.

Variabili: Nomi delle variabili e parole riservate

Esempio di messaggi di errore di sintassi:

```
>>> 76strumenti = "grande banda"
SyntaxError: invalid syntax
>>> milione$ = 1000000
SyntaxError: invalid syntax
>>> class = "Computer Science 101"
SyntaxError: invalid syntax
```

76strumenti è illegale perché non inizia con una lettera

milione$ è illegale perché contiene un carattere non valido (il segno di dollaro $)

class è una delle parole riservate di Python. Le parole riservate definiscono le regole del linguaggio e della struttura e *non possono essere usate come nomi di variabili*.

Variabili: Nomi delle variabili e parole riservate

Parole riservate (keywords):

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

Numeri

I numeri in Python sono di tre tipologie diverse:

- int
- float
- complex

```
>>> x = 1     # int
>>> y = 2.8   # float
>>> z = 2+1j  # complex
```

Il tipo int, o numero intero, è un numero intero, positivo o negativo, senza cifre decimali, di lunghezza illimitata.

Il tipo float, o "numero in virgola mobile" è un numero positivo o negativo, contenente uno o più decimali, secondo lo standard IEEE-754, a 64 bit.

Il tipo complex rappresenta un numero immaginario nella forma a+bj, con j unità immaginaria

Numeri: Rappresentazione degli interi

È possibile rappresentare gli interi in diverse basi (decimale, binario, esadecimale, ottale, ecc..)

L'assegnazione può avvenire in diverse basi, ma l'output della console di Python sarà sempre in base 10.

Es:

```
>>> x = 14          # base 10
>>> x = 0xA0F       # esadecimale (base 16) – prefisso 0x
>>> x = 0o372       # Ottale (base 8) – prefisso 0o
>>> x = 0b1111      # Binario (base 2) – prefisso 0b
>>> x = int("A1B", 12) # Base 12 – funzione int(num, base)
```

Numeri: Operazioni aritmetiche sugli interi

Operazione	Simbolo	Esempio	Risultato
Addizione	+	28 + 10	38
Sottrazione	-	28 - 10	18
Moltiplicazione	*	28 * 10	280
Divisione (intera)	//	28 // 10	2
Modulo (resto)	%	28 % 10	8
Esponente (potenza)	**	28 ** 10	296196766695424

Si noti che tutte queste operazioni sono operazioni con interi.

Ecco perché il risultato di 28 // 10 è 2 e non 2.8

Numeri: Operazioni di confronto sugli interi

Operatore	Descrizione	Esempio
==	*Se i valori di due operandi sono uguali, la condizione diventa vera.*	*(a == b) non è vero.*
! =	*Se i valori di due operandi non sono uguali, la condizione diventa vera.*	*(a != b) è vero.*
<>	*Se i valori di due operandi non sono uguali, la condizione diventa vera.*	*(a <> b) è vero.*
>	*Se il valore dell'operando sinistro è maggiore del valore dell'operando destro, allora la condizione diventa vera.*	*(a > b) non è vero.*
<	*Se il valore dell'operando sinistro è inferiore al valore dell'operando destro, allora la condizione diventa vera.*	*(a < b) è vero.*
> =	*Se il valore dell'operando di sinistra è maggiore o uguale al valore dell'operando di destra, la condizione diventa vera.*	*(a >= b) non è vero.*
<=	*Se il valore dell'operando di sinistra è minore o uguale al valore dell'operando di destra, la condizione diventa vera.*	*(a <= b) è vero.*

Numeri: Operazioni sugli interi

NOTA:

Alcuni altri linguaggi (es. C, Java) memorizzano ciascun numero intero in una piccola quantità di memoria fissa. Questo limita la dimensione del numero intero che può essere memorizzato. I limiti comuni sono 2^{8}, 2^{16}, 2^{32} e 2^{64}.

Python non ha limiti fissi e può memorizzare interi sorprendentemente grandi come $2^{1000000}$ a condizione che sulla macchina sia presente sufficiente memoria e potenza di elaborazione.

Numeri: Priorità delle operazioni

Un'altra cosa importante da tenere a mente è la precedenza degli operatori.

Ad esempio, 1 + 2 // 3 significa (1 + 2) // 3 o 1 + (2 // 3) ?

Python ha un modo specifico e prevedibile per determinare l'ordine in cui esegue le operazioni.

Per le operazioni su interi, il sistema prima gestirà le parentesi () , quindi ** , quindi * , // e % e infine + e - .

Se un'espressione contiene più operazioni che hanno lo stesso livello di precedenza, come * // % , verranno eseguite nell'ordine, da sinistra a destra (per gli operatori associativi a sinistra) o da destra a sinistra (operatori associativi a destra).

Tutti questi operatori aritmetici sono associativi a sinistra, ad eccezione di ** , che è associativi a destra.

Numeri: Priorità delle operazioni

Espressione	Come valuta Python	Risultato
20 + 10 // 2	20 + (10 // 2)	25
20 + 10 - 2	(20 + 10) - 2	28
20 - 10 + 2	(20 - 10) + 2	12
20 - 10 * 2	20 - (10 * 2)	0
20 // 10 * 2	(20 // 10) * 2	4
20 * 10 // 2	(20 * 10) // 2	100
20 * 10 ** 2	20 * (10 ** 2)	2000

Numeri: Priorità delle operazioni

Sr.No.	Operatore	Descrizione
1	**	Exponentiation (raise to the power)
2	~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
3	* / % //	Multiply, divide, modulo and floor division
4	+ -	Addition and subtraction
5	>> <<	Right and left bitwise shift
6	&	Bitwise 'AND'
7	^ |	Bitwise exclusive `OR' and regular `OR'
8	<= < > >=	Comparison operators
9	<> == !=	Equality operators
10	'= %= /= //= -= += *= **=	Assignment operators
11	is is not	Identity operators
12	in not in	Membership operators
13	not or and	Logical operators

Numeri: Operazioni sui Float

I numeri in virgola mobile (tipo float) sono numeri con un punto decimale o un esponente (o entrambi). Gli esempi sono 5.0, 10.24, 0.0, 12. e .3. Possiamo usare la notazione scientifica per indicare numeri in virgola mobile molto grandi o molto piccoli, ad es. 3.8×10^{15}. La prima parte del numero, 3.8, è la mantissa e 15 è l'esponente. Possiamo pensare all'esponente come il numero di volte in cui dobbiamo spostare il punto decimale a destra per ottenere il valore effettivo del numero.

In Python, possiamo scrivere il numero 3.8×10^{15} come 3.8e15 o 3.8e+15. Possiamo anche scrivere come 38e14 o .038e17: hanno tutti lo stesso valore.
Un esponente negativo indica numeri più piccoli, ad es. 2.5e-3 è uguale a 0.0025. Gli esponenti negativi possono essere pensati come quante volte dobbiamo spostare il punto decimale a sinistra. La mantissa negativa indica che il numero stesso è negativo, ad es. -2.5e3 è uguale a -2500 e -2.5e-3 è uguale a -0.0025.

Numeri: Operazioni aritmetiche sui float

Operazione	Simbolo	Esempio	Risultato
Addizione	+	28.56 + 10.4	38.96
Sottrazione	-	28.56 – 107.7	-79.14
Moltiplicazione	*	28.34 * 11.75	332.995
Divisione	/	28.0 / 25.0	1.12
Esponente (potenza)	**	28.0 ** 1.2	54.52424211444941

Altre funzioni matematiche che operano sui numeri float sono disponibili nel modulo math.

NB: i risultati sono approssimati secondo le specifiche delle operazioni sui float IEEE-754 64bit

Numeri: Operazioni di confronto sui float

Operatore	Descrizione	Esempio
==	*Se i valori di due operandi sono uguali, la condizione diventa vera.*	*(a == b) non è vero.*
! =	*Se i valori di due operandi non sono uguali, la condizione diventa vera.*	*(a != b) è vero.*
<>	*Se i valori di due operandi non sono uguali, la condizione diventa vera.*	*(a <> b) è vero.*
>	*Se il valore dell'operando sinistro è maggiore del valore dell'operando destro, allora la condizione diventa vera.*	*(a > b) non è vero.*
<	*Se il valore dell'operando sinistro è inferiore al valore dell'operando destro, allora la condizione diventa vera.*	*(a < b) è vero.*
> =	*Se il valore dell'operando di sinistra è maggiore o uguale al valore dell'operando di destra, la condizione diventa vera.*	*(a >= b) non è vero.*
<=	*Se il valore dell'operando di sinistra è minore o uguale al valore dell'operando di destra, la condizione diventa vera.*	*(a <= b) è vero.*

Stringhe: Rappresentazione delle stringhe

Una stringa è una sequenza di caratteri Unicode.

Le stringhe sono tra i tipi più popolari in Python. Possiamo crearli semplicemente racchiudendo i caratteri tra virgolette (‘ ’ - apice singolo, oppure “ ” - apice doppio).

Python tratta le virgolette singole come le virgolette doppie. La creazione di stringhe è semplice come assegnare un valore a una variabile. Ad esempio:

```
>>> var1 = 'Hello World!'
>>> var2 = "Python3 Programming"
```

Stringhe: lunghezza di una stringa

Per conoscere la lunghezza in caratteri di una stringa è possibile utilizzare la funzione len(*stringa*).

Ad esempio:

```
>>> var1 = 'Hello World!'
>>> len(var1)
12
```

Stringhe: slicing

Python non supporta un tipo "carattere", come, ad esempio, nel C il tipo char; questi sono trattati come *stringhe di lunghezza uno*, e quindi può anche essere considerato una sottostringa.
Per accedere o estrarre le sottostringhe, si utilizzano le parentesi quadre a destra del nome della stringa di partenza, con due indici, che stanno a indicare da quale carattere della stringa partire e a quale carattere fermarsi. Se si utilizza un solo indice, verrà estratto un unico carattere dalla posizione richiesta. Ad esempio:

```
var1 = 'Hello World!'
var2 = "Python Programming"
print("var1[0]: ", var1[0])
print("var2[1:5]: ", var2[1:5])
```

OUTPUT

RUN

```
var1[0]:  H
var2[1:5]:  ytho
```

Stringhe: slicing

È possibile "aggiornare" (perdita di riferimento in memoria) una stringa esistente riassegnandole un nuovo valore o assegnandole il valore di un'altra stringa o sottostringa. Il nuovo valore può essere correlato al suo valore precedente o ad una stringa completamente diversa. Ad esempio:

```
var1 = 'Hello World!'
var2 = var1[:6] + 'Python!'
print("Nuova stringa:", var2)
```

OUTPUT

RUN

```
Nuova Stringa: Hello Python!
```

Stringhe: slicing e operatori

Esempio: sia a = "Hello" e b = "Python"

Operatore	Descrizione	Esempio
+	Concatenazione - Aggiunge valori su entrambi i lati dell'operatore	a + b darà HelloPython
*	Ripetizione: crea nuove stringhe, concatenando più copie della stessa stringa	a * 2 darà -HelloHello
[]	Slice: fornisce il carattere dall'indice specificato	a [1] darà e
[:]	Range Slice - Fornisce i caratteri dell'intervallo specificato	a [1: 4] darà ell
in	Appartenenza - Restituisce vero se un carattere esiste nella stringa specificata	H in a darà 1
not in	Appartenenza - Restituisce vero se un carattere non esiste nella stringa specificata	M non in a darà 1
r / R	Raw String - Sopprime il significato effettivo dei caratteri di escape. La sintassi per le stringhe non elaborate è esattamente la stessa delle stringhe normali con l'eccezione dell'operatore di stringa non elaborato, la lettera "r" che precede le virgolette. La "r" può essere in minuscolo (r) o maiuscolo (R) e deve essere posta immediatamente prima del primo segno di virgoletta.	r '\ n' stampa \n

Stringhe: slicing

Ulteriori esempi:

```
>>> s = "0123456789"

>>> s[:5]      # I primi 5 caratteri

'01234'

>>> s[-5:]     # Gli ultimi 5 caratteri

'56789'

>>> s[1:-1]    # Tutti i caratteri tranne il primo e l'ultimo
'12345678'

>>>
```

Stringhe: slicing

C'è ancora un tipo di slicing che non abbiamo esaminato. Possiamo utilizzare un terzo parametro che indica, oltre all'inizio e alla fine della nostra "fetta", ogni quanti elementi dobbiamo estrarne uno. Forse è meglio chiarire il concetto con un esempio:

```
>>>> s = "0123456789"

>>> s[::2]   # Prendi un carattere ogni due

'02468'

>>> s[1::2]  # Questa volta parti dal secondo

'13579'

>>> s[::-1]  # Rovescia la stringa

'9876543210'
```

Stringhe: slicing

Un'importante caratteristica delle stringhe è che sono oggetti immutabili.

Non possiamo assegnare un nuovo valore a una parte di una stringa

```
>>> s = "ab-de"
>>> s[2] = "c"
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'str' object does not support item assignment

>>>
```

Stringhe: slicing

Per poterlo fare dobbiamo riassegnare l'intera stringa

```
>>> s = "ab-de"
>>> s = s[:2] + "c" + s[3:]
>>> s
'abcde'

>>>
```

Stringhe: caratteri di escape

Per inserire in una stringa un carattere di “a capo” dobbiamo utilizzare il simbolo backslash (la barra rovesciata), seguito dal carattere “n”, ovvero la sequenza \n

```
>>> s = "1\n2\n3\nVia!"
>>> print(s)
1
2
3
Via!

>>>
```

La sequenza \n è un simbolo speciale. Ne esistono diversi altri, ma quelli più importanti sono \n Line Feed (a capo), \r Carriage Return (ritorno carrello), \b Backspace (indietro di un carattere), \t Tab (carattere di tabulazione) e \xHH che permette di inserire il simbolo il cui codice ASCII è il valore esadecimale HH.

Stringhe: concatenamento e ripetizione

Per concatenare tra di loro due stringhe possiamo usare il simbolo +

Possiamo perfino moltiplicare una stringa, usando il simbolo *

```
>>> s = "Precipite"
>>> p = "volissimevolmente"
>>> s+p
'Precipitevolissimevolmente'

>>>
```

```
>>> s = "ITS"
>>> s * 5
'ITSITSITSITSITS'
>>> (s + " ") * 5
'ITS ITS ITS ITS ITS '

>>>
```

Stringhe: metodi

Inoltre, le stringhe possiedono diversi metodi per la loro gestione, che le rendono oggetti particolarmente versatili e potenti

capitalize	format_map	isnumeric	maketrans	split
casefold	index	isprintable	partition	splitlines
center	isalnum	isspace	replace	startswith
count	isalpha	istitle	rfind	strip
encode	isascii	isupper	rindex	swapcase
endswith	isdecimal	join	rjust	title
expandtabs	isdigit	ljust	rpartition	translate
find	isidentifier	lower	rsplit	upper
format	islower	lstrip	rstrip	zfill

Stringhe: metodi (esempi)

```
>>> s = "Troviamo la x in questa stringa"
>>> s.find("x")
12
>>> s = " ITS "
>>> s.strip()
'ITS'
>>> s = "Non mi piace frequentare l'ITS!"
>>> s.replace("Non", "Ma quanto")
"Ma quanto mi piace frequentare l'ITS!"
>>> s.lower()
"non mi piace frequentare l'its!"
>>> s.upper()
"NON MI PIACE FREQUENTARE L'ITS!"

>>>
```

Stringhe: metodo format

Uno dei metodi più interessanti delle stringhe di Python è format. Questo metodo permette di formattare una stringa o l'output di un programma, in maniera analoga alla famiglia delle funzioni printf di C.

Di seguito qualche semplice esempio:

```
>>> s = "Mi chiamo {0} e peso {1:,} kg.".format("King Kong", 1000)
>>> print(s)
Mi chiamo King Kong e peso 1,000 kg.
>>> s = "{0} è {0:X} in base 16 e {0:b} in base 2".format(255)
>>> print(s)
255 è FF in base 16 e 11111111 in base 2
>>> s = "{0} è {0:04X} in base 16 e {0:016b} in base 2".format(255)
>>> print(s)
255 è 00FF in base 16 e 0000000011111111 in base 2

>>>
```

Input e Output: funzione print

La funzione print serve a "stampare" in output (tipicamente sullo schermo, standard output, ma volendo anche su file o altri stream) il valore di una variabile o di una espressione

```
>>> a = 12
>>> b = 3
>>> print(a, b, a * b)
12 3 36
>>> print("a + b =", a + b)
a + b = 15

>>>
```

Si può notare che la funzione print accetta in input un numero variabile di argomenti (anche di tipi diversi), li converte in stringhe, e li mostra in output separati da uno spazio e seguiti da un carattere di ritorno a capo (\n). print consente inoltre di specificare il separatore, il carattere finale, lo stream, e altri parametri.

Input e Output: funzione input

La funzione input viene usata per consentire all'utente di immettere dati da tastiera, che verranno poi utilizzati dal programma.

Essa accetta un singolo argomento opzionale: una stringa che viene mostrata a video prima di leggere il valore digitato. Una volta che l'utente ha digitato un valore e premuto il tasto Invio, input restituisce il valore come stringa, come mostra il seguente esempio:

```
>>> nome = input("Inserisci il tuo nome: ")
Inserisci il tuo nome: >? ITS
>>> print(nome)
ITS

>>>
```

In questo esempio possiamo vedere che:

1) la stringa passata a input che chiede di inserire il nome viene mostrata a video;
2) input attende che l'utente digiti il nome (ITS) e prema Invio;
3) input restituisce il nome inserito come stringa;
4) la stringa restituita da input viene assegnata alla variabile nome;
5) la variabile nome può poi essere utilizzata per accedere al nome inserito dall'utente.

Input e Output: funzione input

Se si desidera immettere valori numerici o di altri tipi, è possibile convertire la stringa restituita da input usando funzioni come int o float

```
>>> raggio = input('Inserisci il raggio: ')
Inserisci il raggio: >? 8.4
>>> r = float(raggio)
>>> raggio
'8.4'
>>> r
8.4
>>> type(raggio), type(r)
(<class 'str'>, <class 'float'>)
>>> print('Area:', 3.14 * r ** 2)
Area: 221.5584
>>> print('Circonferenza:', 2 * 3.14 * r)
Circonferenza: 52.752

>>>
```

In questo esempio possiamo vedere che:

1) input chiede all'utente di inserire un raggio che viene assegnato alla variabile raggio;
2) la funzione float viene usata per convertire raggio (che è una stringa) in un valore numerico con la virgola che viene assegnato alla variabile r;
3) le due variabili (raggio e r) vengono stampate per mostrare che la prima è una stringa (quindi rappresentata tra '...') mentre la seconda è un numero;
4) r viene poi usata per calcolare l'area del cerchio (** è l'operatore di elevazione a potenza) e la funzione print viene usata per stampare la stringa 'Area:' e il risultato dell'espressione 3.14 * r ** 2;
5) r viene usata anche per calcolare la circonferenza e print per stampare la stringa 'Circonferenza:' e il risultato dell'espressione 2 * 3.14 * r;

ESERCIZI

Esercizio 1. Dati il numero intero 1 ed il reale 1.0, quali sono i metodi messi a disposizione dai numeri interi? Dai numeri reali?

Esercizio 2. Data una stringa x = "testo", aprire il manuale del metodo **replace** delle stringhe.

Esercizio 3. Creare alcune variabili: a e b di valore intero 12 e 23, c e d di valore intero lungo 34 e 45, x e y di valore 21.0 e 14.0. Creare un'altra variabile pi di valore 3.141593. Di che tipo è?

Esercizio 4. Usando **print**, stampare a schermo la variabile a. Stampare a e b, sulla stessa riga. Stampare a e b sulla stessa riga, separate da un punto e virgola.

Esercizio 5. Usando **print**, stampare a schermo il prodotto di a e b. Assegnare ad r il risultato del prodotto di a e b.

ESERCIZI

Esercizio 6. Determinare il valore ed il tipo di

1. Il prodotto di a e b
2. La differenza di c e d
3. Il quoziente di x e y
4. Il quoziente intero di a e b
5. Il quoziente intero di c e d
6. Il quoziente intero di x e y
7. Il prodotto di a e c
8. Il prodotto di b e y
9. Il prodotto di x e d
10. 2 elevato alla 107
11. 2 elevato alla 100?
12. 2 elevato alla 1.2?
13. 2 elevato alla -2?
14. La radice quadrata di 4? (Si usi **)

ESERCIZI

Esercizio 7. Cosa succede (in termini di valore e tipo del risultato) se eseguo:

`10 / 12 ; 10 / 12.0 ; 10 // 12 ; 10 // 12.0 ; 10 % 3 ; 10 % 3.0`

Esercizio 8. Usando la costate π approssimata pi = 3.1415926536 e dato r = 2.5, calcolare:

1. La circonferenza di raggio r: circ = $2\pi r$
2. L'area di un cerchio di raggio r: area = πr^2?
3. Il volume di una sfera di raggio r: vol = $4/3*\pi r^3$.

Il linguaggio Python (ver. 3)

Bibliografia

- **Marco Beri, Programmare in Python, Apogeo**
- **Kenneth A. Lambert, Programmazione in Python, Apogeo**
- **Python Software Foundation, https://docs.python.org/3/**
- **Python - sito ufficiale della comunità italiana, www.python.it/doc/**
- **W3C programming language tutorials, www.w3schools.com/python/**
- **“The Hitchhiker’s Guide to Python!”, Kenneth Reitz, Tanya Schlusser, 2018,**
  - https://media.readthedocs.org/pdf/python-guide/latest/python-guide.pdf
- **www.tutorialspoint.com**
  - www.tutorialspoint.com/python3/index.htm

Bibliografia

Software (free license)

- Interprete Python3: www.python.org/downloads/
- IDE: PyCharm (community): www.jetbrains.com/pycharm/download/

Immagini (free license)

- Fonte: www.duckduckgo.com