Riccardo Polignieri

Python in Windows

gli strumenti, le tecniche e l'ambiente di lavoro

Riccardo Polignieri

Python in Windows

gli strumenti, le tecniche e l'ambiente di lavoro

versione 2 – ottobre 2020

Questo libro è in vendita su Leanpub: https://leanpub.com/pythoninwindows

Aggiornamenti

nella versione 2 – ottobre 2020:

- un nuovo capitolo “Librerie esterne utili in Windows”;
- un nuovo capitolo “Scrivere codice cross-piattaforma”;
- revisioni e immagini nuove per aggiornare il libro a Python 3.9.0;
- nel capitolo “Localizzazione e internazionalizzazione”, menzionato il nuovo comportamento di Python 3.8 con il locale in Windows;
- nel capitolo “Editor, IDE e shell interattive”, un nuovo paragrafo su Repl.it;
- nello stesso capitolo, consigliato Pylance invece di Intellicode per Visual Studio Code.

INDICE

CAPITOLO 1

INTRODUZIONE

Python è ormai da molti anni uno dei linguaggi di programmazione più popolari, adottati, insegnati. Ha raggiunto uno status di trasversalità e diffusione di cui solo pochi altri godono: con l'ulteriore vantaggio di essere percepito come più «semplice», facile da usare, amichevole addirittura. Il risultato è che oggi Python è probabilmente l'unico linguaggio a essere usato in modo paritario sia da tecnici e professionisti *della programmazione*, sia dai professionisti dei campi più disparati. Biologi, medici, astronomi, ingegneri, statistici, ricercatori di ogni disciplina usano Python come uno strumento flessibile e potente per ottenere risultati in tempi rapidi nelle loro professioni.

Ma non sono solo gli ambienti della ricerca scientifica e tecnologica a essere attratti dal linguaggio. Python è uno dei motori principali di una trasformazione sociale che negli ultimi vent'anni ha completamente cambiato il panorama della programmazione, ampliandolo in modo straordinario: l'idea della «programmazione alla portata di tutti», della programmazione per i non-programmatori, per una generazione di *power-user* completamente diversa degli «ingegneri del software», ma anche dagli *hacker* degli anni Settanta e Ottanta del secolo scorso, o dai *nerd* di Linux a cavallo del millennio. A questa platea sterminata di utenti, Python promette (e spesso mantiene) potenza coniugata a universalità d'impiego e semplicità d'uso.

Tuttavia «semplice» è un termine relativo. Python richiede pur sempre una non trascurabile quantità di cerimoniale sintattico, il suo modello di esecuzione è ostico almeno in certe parti (i moduli, i package, il sistema degli import…), il suo ecosistema è tutt'altro che immediatamente comprensibile (i virtual environment, Pip, Setuptools…). Gli aspetti di Python che un programmatore esperto considera «semplici», «eleganti», «espressivi» (la meccanica degli iterabili e degli iteratori, per esempio) a un principiante risultano spesso opachi. Anche la semplicità d'uso è ingannevole. È vero, se azzecchi le cinque righe di codice giuste, con Python puoi implementare un web server: ma se non capisci il significato profondo di quelle cinque righe, non andrai lontano. E se metti un piede fuori dal sentiero del tutorial, se cerchi di adattare un esempio trovato in rete, se provi a «smanettare», per lo più ti ritrovi in una selva di complessità. Dopo tutto, Python non è *magia*: è un linguaggio di programmazione con le sue regole e le sue strutture da capire.

Ma forse, dal punto di vista del principiante, la parte più difficile non è tanto Python-il-linguaggio, ma piuttosto quella zona grigia che gli sta tutto intorno e che costituisce il confine incerto tra il linguaggio e il suo ambiente di esecuzione: ovvero, il nostro computer. Una volta che ho Python «installato», come faccio poi a «usarlo» per le mie necessità di tutti i giorni? Perché quando scrivo `open('miofile.txt')`, talvolta funziona e talvolta no? Perché non funziona fare doppio clic su uno script Python? E quando invece funziona? E se scrivo un modulo Python, dove lo metto? E come

faccio a fargli importare un altro modulo? E come devo usare la shell? Che cosa è l'interprete dei comandi, e che cosa c'entra con Python?

Il problema è che i manuali tradizionali non sono poi molto generosi di informazioni al riguardo. In parte è comprensibile: si concentrano sul compito di insegnare Python-il-linguaggio. Tuttavia un principiante che arriva alla fine di un buon manuale potrebbe sapere molte cose sulle metaclassi, e paradossalmente chiedersi perché non può importare un modulo che risiede in una «cartella» qualsiasi. O non avere la minima idea di come si testa il codice. O della struttura che bisogna dare a un progetto. O di come conviene organizzare un virtual environment. O di che cosa succede se ha due versioni di Python installate nel suo computer. O perché certe volte le «lettere accentate» sono sbagliate.

Questo problema della «zona grigia di confine» è di solito più avvertito dagli utenti Windows. Ci sono molte ragioni storiche: Python è nato e si è evoluto in una comunità di programmatori Unix, e per molti anni il suo utilizzo in Windows è stato fortemente minoritario; la maggior parte dei *core developer* ha meno esperienza con Windows; in rete si trovano molti più esempi, guide, manuali che danno per scontato che il lettore abbia Linux; ancora fino a poco tempo fa il supporto del linguaggio per Windows era difettoso in alcuni punti strategici (e qualche problema rimane, per esempio nel *locale*). Ancora oggi si trovano pacchetti Python che credono sempre di essere eseguiti in Linux, con esiti talvolta buffi, magari perché il loro autore non si è mai posto il dubbio che le cose potrebbero andare diversamente.

A loro volta, gli utenti Windows hanno tradizionalmente meno esperienza con tutto l'ecosistema (e la «mentalità Unix») che ruota intorno a Python e ai linguaggi di programmazione come Python. Questo gap culturale è da sempre oggetto di caricatura: il prototipo dell'utente Windows è quello che chiama «cartella» una directory; che chiede dove deve cliccare; che non sa che cosa è un terminale (vuoi dire la finestrella del DOS?); che per prima cosa si informa se è possibile «compilare in exe» il suo programma, e così via. Va detto che per fortuna la comunità Python ha una tradizione di accoglienza e gentilezza verso i principianti: ma può essere comunque sconfortante provare ad aiutare qualcuno così distate da tutto ciò che in genere si dà per acquisito.

Le cose stanno cambiando velocemente in questi anni. In parte, semplicemente, gli utenti di Python in Windows sono cresciuti molto, a tutti i livelli. Microsoft, dal canto suo, ha fatto sforzi enormi per avvicinarsi al mondo dell'open source e per rendere Windows una piattaforma più adatta ai programmatori: e in effetti, piattaforme come Azure e strumenti come Visual Studio Code riscuotono apprezzamento anche fuori dal mondo Windows. Infine, il supporto di Python è migliorato parecchio: oggi il responsabile delle *release* Windows di Python, Steve Dower, è un dipendente di Microsoft e segue anche lo sviluppo dei «Python Tools for Visual Studio». L'interesse, anche aziendale, di Microsoft per Python è sempre più forte e si vede bene, per esempio, nell'attenzione che Azure dedica al linguaggio.

Tuttavia, a lato pratico, resta ancora molto attrito cognitivo a ostacolare il principiante che si accosta a Python «dal lato Windows», soprattutto se è il suo primo linguaggio di programmazione. Mentre le risorse a disposizione per imparare Python non mancano di certo, quelle per imparare *a usare* Python (su Windows, poi) sono quasi inesistenti.

Questo libro cerca appunto di colmare la lacuna.

Per chi è questo libro.

Questo libro si rivolge soprattutto ai principianti, con la speranza di accompagnarli per un certo tratto del loro percorso di apprendimento e di familiarizzazione con Python.

Tuttavia, questo libro *non è* un manuale di Python. Non insegna le variabili, le funzioni, gli `if` e i `for`, le classi e così via. L'idea è che questo libro si dovrebbe leggere *insieme* a un buon manuale, tenendolo accanto. I primi capitoli dovrebbero essere letti addirittura prima di installare Python. Alcuni altri sono per il principiante che ha già acquisito qualche conoscenza in più. Altri ancora riguardano temi più complessi: non è necessario leggerli subito, ma possono essere tenuti da parte per il futuro; oppure è possibile scorrerli rapidamente, per farsi un'idea dei problemi e per sapere che, comunque, prima o poi dovranno essere affrontati.

A proposito di certi capitoli più avanzati, come quelli sull'internazionalizzazione, sui test, sulla distribuzione e così via: è possibile che questi interessino anche qualche programmatore un poco più esperto, magari alla ricerca del «punto di vista di Windows» sull'argomento. Tuttavia mi sembra onesto avvertire che anche questi capitoli sono scritti con il pubblico dei principianti in mente: gli argomenti non sono approfonditi nei dettagli tecnici, ma presentati e descritti nei loro aspetti più semplici, sia pure con grande attenzione a non *banalizzare* mai i problemi.

E a questo proposito, due avvertimenti conclusivi sono dovuti anche ai principianti. Il primo, questo libro non vuole sostituire la documentazione ufficiale di Python e degli altri strumenti di lavoro: l'intento è presentare ogni argomento e spiegarlo fino al punto in cui il lettore è in grado di proseguire da solo a sperimentare e documentarsi meglio. Ma nulla può sostituire lo studio approfondito della documentazione. In secondo luogo, qui non troverete ricette da copiare e incollare per risolvere il vostro problema. L'autore di questo libro ha sviluppato nel tempo una forte e motivata antipatia per le «ricette», gli «esempi pratici», gli «snippet» di codice da adattare. Il *learn by doing* è senz'altro un pilastro irrinunciabile della pedagogia: ma funziona in modo appropriato solo nel contesto di un rapporto di insegnamento diretto. Cercare di applicarlo in un libro, a colpi di esempi, tutorial e «casi di studio», ha un sorprendente effetto negativo sulla qualità dell'insegnamento. Induce un falso senso di sicurezza sulla effettiva complessità della materia trattata, e non incoraggia davvero l'esplorazione autonoma. Beninteso, alcuni libri di «ricette» sono davvero ottimi e hanno una gloriosa tradizione, a partire dal *Python Cookbook*: questo libro, nel suo piccolo, preferisce un'altra strada.

Di che cosa parla questo libro.

Dopo queste premesse, riassumiamo rapidamente il contenuto dei singoli capitoli.

Dopo questa introduzione, il secondo e il terzo capitolo, **Prima di installare Python**, presentano l'ambiente di esecuzione che dovrà accogliere i programmi scritti in Python. Qui parliamo dell'interprete dei comandi, del file system e della directory corrente, di come si invocano i programmi dall'interprete, delle variabili d'ambiente e della path di sistema, delle associazioni dei file e altro ancora. Insomma, tutto ciò che serve a prepararsi a usare Python nell'ambiente di Windows.

Il quarto capitolo, **Installazione di Python**, descrive come installare Python. Ci sono diverse opzioni possibili e prendiamo in considerazione le tre più comuni: l'installer tradizionale, Anaconda e il pacchetto del Microsoft Store.

Il capitolo seguente, **Una prima panoramica di Python**, presenta la shell di Python: dapprima invocata direttamente nella finestra della console, poi più comodamente nell'ambiente grafico di Idle. Forniamo anche le informazioni essenziali sui moduli: come si scrivono, come si eseguono.

Il sesto capitolo offre **Una panoramica della libreria standard**: presentiamo alcune delle funzionalità più comuni di Python, soprattutto dal punto di vista del rapporto con il sistema operativo Windows. Parliamo delle operazioni con il file system, delle informazioni sulla piattaforma e l'architettura del sistema e così via.

Segue un capitolo su come **Installare più versioni di Python**: non è un'esigenza immediata del principiante, ma presto o tardi il problema si porrà da solo, quando cercherà di aggiornare Python a una nuova versione. Qui parliamo anche del *launcher* `py.exe` e del suo uso. Questo capitolo conclude una ideale prima parte introduttiva del libro.

L'ottavo capitolo riguarda i **Virtual environment**: qui iniziamo a parlare di come Python deve essere usato nella pratica comune di programmazione. I «venv» sono un componente essenziale dell'ecosistema e devono essere compresi e utilizzati quanto prima possibile.

A questo segue, naturalmente, un capitolo su come **Installare pacchetti esterni**. Qui parliamo della *repository* ufficiale PyPI, del *packet manager* Pip, del suo uso nei virtual environment, delle liste di *requirement*: tutto l'occorrente per gestire la «casa» del nostro programma Python.

Facciamo quindi una rassegna di **Librerie esterne utili in Windows**: parliamo di come accedere alle API di Windows, interfacciarsi con le applicazioni Office e altro ancora; chiudiamo con una panoramica sulle librerie più popolari in diversi ambiti applicativi.

Segue un breve capitolo a sé stante: **Uso della shebang** parla di come fare i conti con una consuetudine del mondo Linux che potrebbe disorientare l'utente Windows.

Ma la cosa che più di tutte disorienta il principiante in Windows, è che Python è un programma «a riga di comando», senza interfaccia grafica. Nel dodicesimo capitolo parliamo quindi di **Applicazioni Win32 host**, della differenza tra queste e le applicazioni *console host*, e di come rendere Python un'applicazione Win32 all'occorrenza (e di tutte le trappole connesse).

Il tredicesimo capitolo è dedicato a **Unicode**, fonte inesauribile di fraintendimenti, e non solo tra i principianti. Iniziamo con una lunga panoramica sul tema, quindi percorriamo il «triangolo» di argomenti: come funziona Unicode in Windows, come funziona Unicode in Python, come funziona Unicode in Python su Windows. Questo capitolo completa una seconda parte del libro, dedicata ad alcuni aspetti più tecnici sull'uso di Python in Windows.

Il capitolo successivo inaugura una terza parte, dedicata alla «vita quotidiana del programmatore Python». Partiamo dalla **Struttura di un progetto Python** per descrivere come può essere organizzato il codice; per inquadrare il problema, parliamo prima brevemente del sistema degli `import`; mostriamo come localizzare correttamente le risorse esterne che non sono dei moduli; infine diamo delle indicazioni su questioni di stile e uso dei *linter*.

Dedichiamo poi due capitoli al complicato tema di come **Distribuire il codice** che abbiamo scritto. Presentiamo Setuptools e seguiamo tutte le fasi della pacchettizzazione di un progetto Python, fino alla pubblicazione su PyPI. Esaminiamo quindi alcune altre **Opzioni di distribuzione per Windows**, con particolare attenzione agli archivi eseguibili, e concludendo con una panoramica dei vari *tool* per produrre distribuzioni *stand-alone*.

Ci occupiamo quindi di **Localizzazione e internazionalizzazione**, un tema spesso trascurato e sottovalutato anche dai professionisti. Presentiamo una panoramica sull'argomento, con la dovuta attenzione ai problemi specifici che si possono incontrare in ambiente Windows.

Il capitolo seguente parla degli ormai onnipresenti sistemi per il **Controllo di versione**: da strumento iper-specializzato di interesse solo per i grandi team di sviluppo, i VCS sono diventati curiosamente popolari soprattutto come modo di «socializzare» il codice. Qui presentiamo in modo molto sintetico Git per Windows, descriviamo come pubblicare una repository su GitHub e come sottoporre una *pull request* per collaborare a un progetto altrui.

Testare il codice è il mantra dello sviluppo con Python: anche se non c'è molto da dire su questo tema dal punto di vista specifico di Windows, ci sembra comunque importante presentare le tecniche di base dei test con `unittest`. Parliamo anche di indice di copertura e di automazione con Tox.

L'altro principio guida della comunità Python è che bisogna sempre **Documentare il codice**. Parliamo di come si scrivono le *docstring*, di come generare la documentazione con Sphinx e di come pubblicarla su ReadTheDocs attraverso una repository Git.

Aggiungiamo infine alcuni brevi consigli su come **Scrivere codice cross-piattaforma**: anche se noi lavoriamo in Windows, dobbiamo comunque fare attenzione a non escludere gli utenti di altri sistemi operativi dal nostro codice.

Chiudiamo con una panoramica su **Editor, IDE e shell interattive**. La scelta di un editor è sempre una questione delicata e personale per i programmatori esperti; per i principianti è di solito un salto nel buio, confusi tra le decine di opzioni. Proviamo quindi di inquadrare l'argomento in generale e ci soffermiamo poi brevemente su Vim, SublimeText, Visual Studio Code e gli Jupyter Notebook.

1.1 La comunità di Python.

Chiudiamo questa introduzione con alcune note di contesto sulla *governance* di Python e sulla sua comunità.

Python è un linguaggio di programmazione presentato da Guido van Rossum per la prima volta nel 1991. Dopo alcuni passaggi, a partire dal 2001 la licenza di Python è detenuta dalla Python Software Foundation, un'organizzazione non-profit. A partire da quella data, e dalla versione 2.1 del linguaggio, la licenza di Python è GPL-compatibile ma più permissiva della GPL: è possibile anche modificare Python e distribuirlo come software proprietario.

La Python Software Foundation si occupa di promuovere lo sviluppo del linguaggio, di raccogliere fondi, di mantenere l'infrastruttura dei siti web e della *package repository* ufficiale PyPI, e così via; organizza eventi e conferenze, a partire dalla PyCon annuale.

Guido van Rossum ha guidato il gruppo dei *core developer* del linguaggio fino al 2018 nella sua qualità di *Benevolent Dictator For Life*; da quando ha deciso di ritirarsi dalla posizione, è stato discusso e approvato un nuovo modello di *governance*. Attualmente il gruppo dirigente di Python è costituito da uno *Steeering council* di cinque persone, che vengono elette dai *core developer* attivi. Il comitato viene rinnovato a ogni nuova *release* di Python.

La repository ufficiale del linguaggio è stata per molti anni governata dal VCS Mercurial, ma dal 2017 Python è migrato a Git e la repository è adesso pubblicata su GitHub. Il processo di sviluppo del linguaggio comporta una serie di discussioni pubbliche successive. L'idea per una nuova *feature* o una modifica importante viene di solito presentata e discussa su Python-dev o Python-ideas, le *mailing list* dei *core developer*. Se si forma sufficiente interesse e consenso per la proposta, questa viene formalizzata in una PEP (*Python Enhancement Proposal*) e pubblicata sul sito web di Python. La discussione sulla PEP continua su Python-dev; la PEP può essere modificata e riformulata molte volte nel processo, e alcune PEP restano aperte per anni. L'ultima parola spetta oggi allo *Steering council*, che può approvarla o respingerla. Se la PEP viene approvata, i *core developer* più interessati si occupano di implementarla e inserirla nella successiva *release* del linguaggio. Le PEP completano e integrano la documentazione ufficiale, illustrando molte *feature* nel dettaglio e spiegando le ragioni della loro introduzione e le possibili alternative che sono state scartate.

Naturalmente, la stragrande maggioranza delle modifiche non richiede la stesura di una PEP: i *bugfix* e i piccoli miglioramenti sono semplicemente discussi e implementati nel *bug tracker* di Python.

La «vita sociale» della comunità Python è ovviamente molto ricca di eventi. L'appuntamento più importante è la Python Conference (PyCon) organizzata annualmente negli Stati Uniti dalla Python Software Foundation; oltre a questa, esistono numerose PyCon nazionali, tra cui una in Italia. Ci sono poi innumerevoli seminari, conferenze e appuntamenti periodici di interesse più specifico o regionale: tra i più importanti possiamo ricordare EuroPython, la SciPy Conference dedicata al

mondo scientifico, i vari appuntamenti di PyData e la DjangoCon. Il livello di granularità più fine è costituito dalle centinaia di Python User Group organizzati dagli appassionati in ogni parte del mondo.

Non è possibile anche solo cominciare ad elencare le centinaia di libri, corsi e risorse online dedicati all'apprendimento di Python, come è logico aspettarsi da uno dei linguaggi di programmazione più popolari da oltre vent'anni. Per un orientamento iniziale è possibile consultare il Python Wiki e il forum Python-forum. Le due risorse italiane più frequentate sono i forum di Python.it e ForumPython.it.

Link utili.

- Python: https://www.python.org;
- La repository GitHub di Python: https://github.com/python/cpython;
- La Python Software Foundation: https://www.python.org/psf-landing;
- La PEP 13, che delinea il modello di *governance* di Python: https://www.python.org/dev/peps/pep-0013;
- L'elenco delle PEP (*Python Enhancement Proposal*): https://www.python.org/dev/peps;
- Il *bug tracker* di Python: https://bugs.python.org;
- Python-dev, la *mailing list* per i *core developer* di Python: https://mail.python.org/archives/list/python-dev@python.org;
- Python-ideas, per discutere nuove idee sul linguaggio: https://mail.python.org/archives/list/python-ideas@python.org;
- Python-list (ovvero comp.lang.python), la lista di discussioni generali: https://mail.python.org/mailman/listinfo/python-list;
- I nuovi forum Discourse che nel tempo dovrebbero sostituire le vecchie liste: https://discuss.python.org;
- La *Developer's Guide* contiene molte informazioni utili sul processo di sviluppo di Python, e su come contribuire: https://devguide.python.org;
- PyCon: https://pycon.org; la PyCon americana: https://us.pycon.org; quella italiana: https://pycon.it; EuroPython: https://europython.eu; la SciPy Conference: http://conference.scipy.org; PyData: https://pydata.org; DjangoCon: https://djangocon.us; un elenco di Python User Group: https://wiki.python.org/moin/LocalUserGroups;
- Il wiki di Python: https://wiki.python.org; Python-forum: https://python-forum.io; ForumPython.it: https://forumpython.it; Python.it: https://www.python.it.