Εισαγωγή στην Python για ερευνητές ανθρωπιστικών επιστημών

Contents

Βασικές έννοιες στην Python

Dr. W.J.B. Mattingly

Μετάφραση: Δρ. Τάσος Κολυδάς

1. Εισαγωγή στην Python

Θέματα που Καλύπτονται σε αυτό το Κεφάλαιο

1. Γιατί οι ερευνητές των ανθρωπιστικών επιστημών πρέπει να μάθουν προγραμματισμό;
2. Γιατί να χρησιμοποιούμε ειδικά την Python;
3. Τι καλύπτεται σε αυτό το εγχειρίδιο;

4. Πώς να χρησιμοποιήσετε την Python από αυτό το Εγχειρίδιο
5. Πώς να χρησιμοποιήσετε την Python στο JupyterOrg
6. Πώς να εγκαταστήσετε την Python με το Anaconda Navigator και το Jupyter-Lab

1.1. Εισαγωγή στην Python

1.1.1. Γιατί Θα Έπρεπε οι Ερευνητές των Ανθρωπιστικών Επιστημών να Μάθουν Προγραμματισμό;

Πριν ξεκινήσουμε αυτό το βιβλίο, ας ξεκινήσουμε με ένα απλό ερώτημα. *Γιατί θα έπρεπε οι επιστήμονες των ανθρωπιστικών επιστημών να μάθουν προγραμματισμό;* Για να το απαντήσουμε, ας εξετάσουμε ένα ιδανικό έργο ψηφιακών ανθρωπιστικών σπουδών.

Σε ένα ιδανικό έργο ψηφιακών ανθρωπιστικών σπουδών, ένας ερευνητής (ή ομάδα ερευνητών) θα αξιοποιεί την **γνώση του πεδίου του** ή το εμπειρικό του πεδίο. Ως ειδικός στο πεδίο του, αυτός ο υπεύθυνος του έργου, γνωστός καλύτερα ως κύριος ερευνητής (PI), θα μεταφέρει την ιδέα του στην ομάδα του, η οποία θα αποτελείται από άλλους στο πεδίο του και συνήθως από έναν τεχνικό ηγέτη. Αυτός ο τεχνικός ηγέτης θα χειριστεί (ανάλογα με τη χρηματοδότηση) όλες τις τεχνικές πτυχές του έργου (ανάπτυξη ιστοσελίδων, προγραμματισμό, διαχείριση δεδομένων κ.λπ.) ή θα ηγηθεί μιας ομάδας για να χειριστεί όλες τις τεχνικές πτυχές. Αυτό είναι το ιδανικό σενάριο. Κάθε άτομο στην ομάδα κάνει αυτό που κάνει καλύτερα.

Ένα τέτοιο έργο απαιτεί δύο πράγματα που είναι σε υψηλή ζήτηση από τους περισσότερους ερευνητές: χρηματοδότηση και χρόνο. Ο αυτάρκης ερευνητής που μπορεί να αξιοποιήσει τη γνώση του πεδίου του και ταυτόχρονα να λειτουργήσει με τεχνική ικανότητα σε ένα έργο μειώνει δραστικά και τα δύο αυτά προβλήματα με διάφορους τρόπους. Για να καταλάβουμε πώς, πρέπει πρώτα να καταλάβουμε πώς λαμβάνουν χρηματοδότηση τα έργα ψηφιακών ανθρωπιστικών σπουδών.

Η μάθηση προγραμματισμού δίνει στον ερευνητή των ανθρωπιστικών επιστημών εργαλεία πέρα από τη δυνατότητα μόνο προγραμματισμού. Αλλάζει θεμελιωδώς τον τρόπο που ο ερευνητής βλέπει το πεδίο του και τα ερωτήματα που μπορεί να εξερευνήσει. Πράγματα που φαίνονταν αδύνατα πριν, θα φαίνονται ξαφνικά αρκετά απλά να λυθούν.

Ένα από τα μεγαλύτερα πλεονεκτήματα της μάθησης προγραμματισμού είναι η **αυτοματοποίηση**, ή η διαδικασία με την οποία γράφουμε κανόνες για ένα υπολογιστικό σύστημα να εκτελέσει μια επαναλαμβανόμενη εργασία. Ως ερευνητές των ανθρωπιστικών επιστημών, κάνουμε πολλά

πράγματα επαναληπτικά. Πολλές από τις εργασίες που πρέπει να εκτελέσουμε είναι επαναλαμβανόμενες και εμείς, ως άνθρωποι, είμαστε επιρρεπείς στο να κάνουμε λάθη. Η ικανότητα αυτοματοποίησης αυτών των εργασιών μπορεί να μειώσει δραστικά τον χρόνο που ξοδεύουμε στην εκτέλεση επαναλαμβανόμενων εργασιών, από ώρες, εβδομάδες ή ακόμη και χρόνια, σε δευτερόλεπτα, λεπτά ή ώρες. Φανταστείτε ότι πρέπει να λάβετε δεδομένα από ένα αρχείο ιστοσελίδας. Φανταστείτε ότι οι πληροφορίες που χρειάζεστε βρίσκονται σε 2.000 διαφορετικές σελίδες. Πόσο χρόνο θα χρειαστεί να πάτε σε κάθε μία από αυτές τις σελίδες και να αντιγράψετε και να επικολλήσετε το κείμενο σε ένα έγγραφο Word; Στην Python, η εργασία μπορεί να κωδικοποιηθεί σε λεπτά και να αφεθεί να εκτελεστεί για μια ώρα. Αυτό σας επιτρέπει, ως ερευνητή, να πάτε και να κάνετε κάποια άλλη πιο απαραίτητη εργασία· ή, ίσως, να απολαύσετε ένα ωραίο τσάι σε μια αιώρα ώστε όταν επιστρέψετε για να αναλύσετε τα έγγραφα, θα είστε ανακεφαλαιωμένοι.

Ο προγραμματισμός δεν μας επιτρέπει απλώς να αυτοματοποιούμε εργασίες όπως αυτή. Μας επιτρέπει επίσης να καθαρίζουμε συστηματικά δεδομένα. Φανταστείτε ότι θέλατε να αναζητήσετε σε διαθέσιμες σαρώσεις PDF μεσαιωνικής λατινικής. Αυτό το πρόβλημα παρουσιάζει πολλά βασικά ζητήματα που καθιστούν μια τέτοια εργασία αδύνατη ή αναξιόπιστη. Πρώτον, η μεσαιωνική λατινική δεν είχε καμία σταθερή σύμβαση ορθογραφίας. Αυτό σημαίνει ότι ορισμένες σαρώσεις ενδέχεται να έχουν διαφορετικές ορθογραφίες λέξεων. Δεύτερον, τα λατινικά είναι μια ιδιαίτερα κλιτή γλώσσα, που σημαίνει ότι η σειρά των λέξεων δεν είναι σημαντική, παρά η λήξη μιας δεδομένης λέξης. Όταν συνδυάζεται με διαφορετικές ορθογραφίες, αυτό σημαίνει ότι κάθε λέξη μπορεί να αναπαρασταθεί μερικές φορές εκατοντάδες διαφορετικούς τρόπους. Επιπλέον, τα κείμενά σας είναι σαρώσεις. Είναι ακόμη και αυτές οι σαρώσεις αναζητήσιμες; Αν είναι, ήταν το OCR, ή Οπτική Αναγνώριση Χαρακτήρων, ακριβές; Εάν έγινε πριν από το 2015, πιθανότατα δεν είναι. Εάν μετά, τότε οι σαρώσεις μπορεί να είναι σε τόσο κακή κατάσταση που το OCR δεν είναι ακριβές. Επιπλέον, οποιοδήποτε σύστημα OCR θα διατηρήσει αλλαγές γραμμής, που σημαίνει ότι εάν η λέξη-κλειδί που θέλετε να αναζητήσετε είναι χωρισμένη επειδή η λέξη χωρίζεται μεταξύ δύο γραμμών, πρέπει να λάβετε αυτό υπόψη στην αναζήτησή σας. Στη συνέχεια, πρέπει να λάβουμε υπόψη τις σημειώσεις του επεξεργαστή, οι οποίες συχνά σημειώνονται σε αγκύλες, παρενθέσεις και άλλα σύμβολα. Ενώ αυτό το παράδειγμα είναι σίγουρα ένα περίπλοκο, είναι απόλυτα κοινό. Και ενώ χρησιμοποιώ λατινικά για να δείξω ένα μεγαλύτερο ζήτημα με κλιτές γλώσσες, αυτά τα ίδια προβλήματα, ειδικά αυτά γύρω από το OCR, προκύπτουν και με αγγλικά κείμενα. Ο προγραμματισμός μας επιτρέπει να αντιμετωπίσουμε καθένα και καθένα από αυτά τα ζητήματα, κάποια πιο εύκολα από ότι άλλα.

Το ζήτημα της αναζήτησης περιπλέκεται περαιτέρω όταν αυτό πρέπει να γίνει για πολλά έγγραφα ταυτόχρονα. Φανταστείτε εάν αυτά τα ζητήματα προκύψουν σε 5.000 διαφορετικά αρχεία PDF που πρέπει να αναλύσετε. Θα μπορούσατε ρεαλιστικά να εκτελέσετε όλες αυτές τις αναζητήσεις σε 5.000 έγγραφα; Αν θα μπορούσατε, θα ήταν τα αποτελέσματά σας καλά; Για να απαντήσουμε στο πρώτο, ναι, εάν είστε πρόθυμοι να ξοδέψετε μήνες να το κάνετε· για να απαντήσουμε στο δεύτερο, πιθανότατα όχι. Ο προγραμματισμός σας επιτρέπει να αναπτύξετε προγράμματα που εκτελούν όλες αυτές τις εργασίες σε όλα τα 5.000 έγγραφα. Όταν εκτελέσετε μια αναζήτηση, δεν θα απλώς πατήσετε ctrl+f. Θα κωδικοποιήσετε τη δική σας μέθοδο αναζήτησης ώστε η απλή αναζήτησή σας να μπορεί να επιστρέψει πολύ περίπλοκα αποτελέσματα που σχετίζονται με τη διακύμανση στο κείμενο.

Η Python κάνει όλα αυτά δυνατά.

1.1.2. Τι είναι η Python;

Η Python είναι μια γλώσσα προγραμματισμού. Οι γλώσσες προγραμματισμού είναι τρόποι που εμείς, ως άνθρωποι, μπορούμε να γράψουμε εντολές που στη συνέχεια θα εκτελεστούν από έναν υπολογιστή. Υπάρχουν πολλές διαφορετικές γλώσσες προγραμματισμού διαθέσιμες για ερευνητές να επιλέξουν από:

1. C
2. Python
3. JavaScript
4. R
5. HTML (αυτό είναι αμφιλεγόμενο)

Όλες οι γλώσσες προγραμματισμού δεν δημιουργούνται ίσες. Κάποιες είναι καλύτερα χρησιμοποιημένες για την ανάπτυξη λογισμικού, όπως η C· άλλες είναι καταλληλότερες για ανάπτυξη ιστοσελίδων, όπως τα HTML και JavaScript. Και άλλες είναι χαρακτηριστικά σε στατιστική ανάλυση, όπως η R (και η Python). Που αριστεύει η Python; Η Python αριστεύει σε πολλές περιοχές. Ένα πράγμα που τη ξεχωρίζει από άλλες γλώσσες προγραμματισμού για νέους κωδικοποιητές είναι ότι είναι εύκολη στη χρήση και εύκολη στη γραφή.

Είναι σχετικά εύκολη να γράψει σε σχέση με άλλες γλώσσες προγραμματισμού επειδή η σύνταξη, ή ο τρόπος με τον οποίο γράφετε εντολές στον κώδικα, είναι ευθύς. Είναι εύκολη στη ανάγνωση επειδή η Python χρησιμοποιεί αναγκαστική εσοχή. Αυτό σημαίνει ότι τα μπλοκ κώδικα που

μπορούν να είναι δύσκολο να διαβαστούν σε άλλες γλώσσες προγραμματισμού, εντοπίζονται εύκολα στην Python.

Από τη δημιουργία της στις αρχές της δεκαετίας του 1990, η Python έχει αυξηθεί σημαντικά στη δημοτικότητα, η οποία, με τη σειρά της, έχει οδηγήσει σε μια μεγάλη κοινότητα προγραμματιστών και ένα μεγάλο αριθμό βιβλιοθηκών που διατίθενται. Θα μάθουμε για τις βιβλιοθήκες αργότερα σε αυτό το εγχειρίδιο. Προς το παρόν, σκεφτείτε τις βιβλιοθήκες ως μεγάλες ποσότητες κώδικα που έχουν ήδη γραφτεί ώστε να μπορείτε να γράψετε μία γραμμή κώδικα για να εκτελέσετε μια περίπλοκη εργασία που μπορεί να χρειαστεί εκατοντάδες ή χιλιάδες γραμμές κώδικα για να γραφεί.

Σήμερα, η Python είναι μια από τις πιο ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού και θεωρείται η απαραίτητη γλώσσα για ανάλυση κειμένου, μηχανική μάθηση, ανάλυση δεδομένων (παράλληλα με το R), διαδίκτυο scraping, και πολλά άλλα.

Η Python (κατά το χρόνο της συγγραφής) είναι αυτήν τη στιγμή σε έκδοση `3.11.0`. Ας διαχωρίσουμε κάθε έναν από αυτούς τους αριθμούς. Το `3` αναφέρεται στην κύρια γλώσσα Python. Η Python 2 υπάρχει ακόμη (στην πραγματικότητα έρχεται ως προεπιλογή σε όλα τα Mac), αλλά δεν υποστηρίζεται πλέον και σταδιακά αντικαθίσταται από την Python 3. Δεν θα πρέπει να επενδύσετε χρόνο στη μάθηση της Python 2 καθώς χρησιμοποιείται μόνο για υποστήριξη legacy λογισμικού και έχει μεγάλη ποσότητα ζητημάτων ασφαλείας λόγω της καταργημένης κατάστασής της. Είναι σημαντικό να σημειώσουμε αυτό γιατί ορισμένα πράγματα κωδικοποιούνται διαφορετικά μεταξύ των δύο γλωσσών προγραμματισμού. Εάν ψάχνετε βοήθεια σε ένα φόρουμ κωδικοποίησης, όπως το StackOverflow, είναι σημαντικό να γνωρίζετε για αυτές τις διακρίσεις.

Το `11.0` στο `Python 3.11.0`, μας λέει ειδικά ποια έκδοση της Python 3 χρησιμοποιούμε. Ο κώδικας από ορισμένες βιβλιοθήκες δεν είναι συμβατός προς τα πίσω, που σημαίνει ότι ορισμένες βιβλιοθήκες απαιτούν συγκεκριμένη έκδοση της Python, οπότε η κατανόηση αυτής της έννοιας τώρα θα εξοικονομήσει σύγχυση αργότερα. Κάθε φορά που εμφανίζεται μια νέα έκδοση, νέα χαρακτηριστικά είναι διαθέσιμα, οπότε είναι σημαντικό να ενημερώνεστε με τρέχουσες εκδόσεις, αλλά δεν είναι πάντα απαραίτητο.

1.1.3. Γιατί Python;

Για όλους τους παραπάνω λόγους, ενθαρρύνω τους ερευνητές των ανθρωπιστικών επιστημών να μάθουν την Python ως πρώτη γλώσσα προγραμματισμού τους. Είναι μια από τις ευκολότερες γλώσσες να μάθει κανείς, ευθεία να γράψει, και μπορεί να λύσει τα περισσότερα από τα

προγραμματιστικά προβλήματα που ένας ερευνητής ενδέχεται να αντιμετωπίσει. Οι μεγάλες ποσότητες βιβλιοθηκών και οδηγών σημαίνουν ότι υπάρχουν λίγες εργασίες που θα αποδειχθούν αδύνατες να γίνουν.

Εάν έχετε πειστεί για την Python, τότε συνεχίστε να διαβάζετε αυτό το εγχειρίδιο καθώς μαθαίνουμε πώς να το εγκαταστήσουμε και να το χρησιμοποιήσουμε ως ερευνητές στις ανθρωπιστικές επιστήμες.

1.2. Εγκατάσταση της Python

Το πιο δύσκολο πράγμα για την εργασία με την Python (ή οποιαδήποτε γλώσσα προγραμματισμού) είναι να μάθετε πώς να την εγκαταστήσετε σωστά. Γιατί είναι αυτό τόσο δύσκολο; Επειδή η εγκατάσταση της Python διαφέρει δραματικά ανά λειτουργικό σύστημα και έκδοση λειτουργικού συστήματος. Παρακάτω βλέπετε διαφορετικούς τρόπους με τους οποίους μπορείτε να ξεκινήσετε προγραμματισμό Python από τον ευκολότερο στον πιο δύσκολο. Εάν είστε εντελώς νέοι στον προγραμματισμό, συνιστώ απλώς να χρησιμοποιήσετε την πρώτη επιλογή. Στην εμπειρία μου, είναι σημαντικό στις αρχές του προγραμματιστικού ταξιδιού σας να έχετε όσο το δυνατόν λιγότερα εμπόδια. Εάν ενδιαφέρεστε να μπείτε στην Python όσο το δυνατόν γρήγορα, τότε χρησιμοποιήστε τα ελεύθερα διαθέσιμα γραφικά στοιχεία Trinket στην ηλεκτρονική έκδοση αυτού του εγχειριδίου και επιστρέψτε στην εγκατάσταση της Python αργότερα.

1.2.1. Trinket

Σε όλο αυτό το εγχειρίδιο, θα δείτε εφαρμογές Trinket ενσωματωμένες σε κάθε σελίδα. Το Trinket σας επιτρέπει να εξασκήσετε τις δεξιότητες κωδικοποίησής σας χωρίς να εγκαταστήσετε ποτέ την Python. Εάν είστε νέοι στον προγραμματισμό και θέλετε να ξεκινήσετε αμέσως χωρίς να εγκαταστήσετε την Python, συνιστώ να εργαστείτε με το Trinket μέσα στην ψηφιακή έκδοση αυτού του εγχειριδίου.

Αυτά θα σας επιτρέψουν να εξασκηθείτε στο μάθημα ακριβώς μέσα στο ψηφιακό εγχειρίδιο. Οι εφαρμογές Trinket μοιάζουν ως εξής:

```
from IPython.display import IFrame
IFrame('https://trinket.io/embed/python3/3fe4c8f3f4', 700, 500)
```

Θα μπορείτε να γράψετε τον κώδικά σας μέσα σε αυτήν την εφαρμογή Trinket και στη συνέχεια να κάνετε κλικ στο κουμπί αναπαραγωγής στο επάνω μέρος. Αυτό θα εκτελέσει τον κώδικά σας και θα συμπληρώσει τα αποτελέσματα στη δεξιά έξοδο.

1.2.2. Εγκατάσταση της Python τοπικά

Εάν επιθυμείτε να εγκαταστήσετε την Python τοπικά και είναι η πρώτη φορά, υπάρχουν πολλά προβλήματα που μπορούν να προκύψουν. Για αυτόν τον λόγο, τώρα συνιστώ σε όλους τους μαθητές να εγκαταστήσουν μέσω του Anaconda Navigator. Προσθέτει επιπλέον βήματα στη διαδικασία εγκατάστασης, αλλά εξαλείφει το δυνατό για λάθη που θα προκύψουν.

Το Anaconda Navigator είναι μια φιλική προς τον χρήστη διεπαφή που χειρίζεται τη διαδικασία εγκατάστασης για σας. Επιτρέπει επίσης να δημιουργήσετε περιβάλλοντα, τα οποία είναι μικρές περιοχές στον υπολογιστή σας που έχουν μια μοναδική έκδοση της Python και βιβλιοθηκών που είναι εγκατεστημένες. Θα μάθουμε περισσότερα σχετικά με αυτό στο Μέρος 06 όταν εξερευνούμε τις βιβλιοθήκες.

Σε αυτό το κεφάλαιο, θα σας καθοδηγήσω στα βήματα εγκατάστασης του Anaconda Navigator στο μηχάνημά σας, ανεξάρτητα από το λειτουργικό σύστημα.

Όταν γράφουμε κώδικα στην Python, το κάνουμε με λίγους διαφορετικούς τρόπους, ανάλογα με τη χρήση αυτού του κώδικα. Επειδή αυτό είναι ένα εγχειρίδιο και ο κώδικας που γράφω είναι για σκοπούς παρουσίασης, χρησιμοποιώ ένα Jupyter Notebook. Άλλες φορές, ενδέχεται να γράψετε ένα πρόγραμμα σε ένα IDE, ή ένα Integrated Development Environment. Ορισμένα από αυτά περιλαμβάνουν PyCharm, Atom, κ.λπ. Σε άλλες περιπτώσεις, θα χρησιμοποιήσετε την Python στο τερματικό για να εκτελέσετε γρήγορες εργασίες σε δεδομένα σε έναν κατάλογο.

Το Anaconda Navigator αφαιρεί την ανάγκη για σας να μάθετε πώς να το κάνετε όλο αυτό επειδή σας επιτρέπει να εγκαταστήσετε εύκολα το Jupyter-Lab το οποίο λειτουργεί σαν IDE αλλά είναι λίγο πιο συγχωρητικό. Επιπλέον, μπορείτε να καλέσετε περιόδους τερματικού. Ξέρω ότι αυτοί οι όροι δεν έχουν νόημα αυτήν τη στιγμή, αλλά καθώς η κατανόησή σας του προγραμματισμού επεκταθεί, αυτή η παράγραφος θα έχει περισσότερο νόημα. Προς το παρόν, απλώς κατανοήστε ότι το Anaconda Navigator και το Jupyter-Lab (και τα δύο των οποίων εγκαθιστούμε σε αυτό το κεφάλαιο), κάνουν την έναρξη της μάθησης προγραμματισμού πολύ, πολύ απλούστερη.

1.2.3. Συμπέρασμα

Εάν σε κάποιο σημείο της διαδικασίας εγκατάστασης νιώσετε απογοήτευση, μην αποθαρρυνθείτε. Δεν είστε μόνοι και δεν είναι μια απλή ή εύκολη διαδικασία. Αυτοί που προγραμματίζουν χρόνια μπορούν να κάνουν αυτά τα βήματα να φαίνονται απλά, αλλά η αντιμετώπιση προβλημάτων μόνοι σας είναι προκλητική. Είναι σημαντικό να θυμάστε ότι ο προγραμματισμός γενικά είναι προκλητικός. Προσπαθήστε να κοιτάξετε τις προκλήσεις που προκύπτουν ως διασκεδαστικά προβλήματα που πρέπει να επιλυθούν.

1.3. Βασικά Στοιχεία Προγραμματισμού

Πριν ξεκινήσουμε το εγχειρίδιο, θα πρέπει να καλύψουμε μερικές βασικές πτυχές του προγραμματισμού που ο αναγνώστης ενδέχεται να μην γνωρίζει. Αυτοί είναι ορισμένοι από τους πιο απαραίτητους όρους και έννοιες που θα δείτε να χρησιμοποιούνται σε όλο αυτό το εγχειρίδιο. Θυμηθείτε, μπορείτε πάντα να επιστρέψετε σε αυτήν την ενότητα εάν ξεχάσετε έναν από αυτούς τους όρους.

1.3.1. Η Συνάρτηση Print

Το πρώτο πράγμα που μαθαίνει κάθε προγραμματιστής είναι πώς να εκτυπώσει κάτι χρησιμοποιώντας τη γλώσσα προγραμματισμού. Στις περισσότερες μαθήματα, θα δείτε τη φράση “Hello, World!” να χρησιμοποιείται. Σε αυτό το εγχειρίδιο, ας δοκιμάσουμε κάτι λίγο διαφορετικό. Ας πούμε ότι ήθελα να εκτυπώσω “Hello, William”. Μπορούμε να το κάνουμε αυτό στην Python χρησιμοποιώντας τη **συνάρτηση print**. Η συνάρτηση print μας επιτρέπει να εκτυπώσουμε κάποιο κομμάτι δεδομένων. Στην περίπτωσή μας, εκείνο το κομμάτι δεδομένων θα είναι ένα κομμάτι κειμένου, γνωστό ως συμβολοσειρά (δες παρακάτω). Το κείμενό μας είναι “Hello, William”. Το πράγμα που θέλουμε να εκτυπώσουμε πρέπει να βρίσκεται μεταξύ μιας ανοικτής και κλειστής παρενθέσεως. Ας δοκιμάσουμε να εκτελέσουμε την εντολή print στο κελί παρακάτω.

```
print("Hello, William!")
```

```
Hello, William!
```

Όπως μπορούμε να δούμε, ο κώδικας που πληκτρολογήσαμε στο κελί του Jupyter εξέδωσε παρακάτω. Τώρα, είναι δική σας σειρά. Δοκιμάστε να εκτυπώσετε κάτι. Προκειμένου να

εκτυπώσετε κείμενο, θα χρειαστείτε να χρησιμοποιήσετε μια ανοικτή και κλειστή εισαγωγική σημείωση. Θα μάθουμε περισσότερα σχετικά με αυτό καθώς συναντούμε συμβολοσειρές στο επόμενο κεφάλαιο.

▶ **Show code cell source**

Αυτό έχει λειτουργήσει υπέροχα, αλλά τι θα συμβεί αν το σημειωματάριό μας Python πρέπει να είναι πιο δυναμικό, δηλαδή πρέπει να προσαρμοστεί με βάση κάποια είσοδο του χρήστη. Ίσως, πρέπει να χρησιμοποιήσουμε ένα όνομα άλλο από το William με βάση κάποια είσοδο που καθορίζεται από τον χρήστη. Για να το κάνουμε αυτό, θα πρέπει να αποθηκεύσουμε ένα κομμάτι δεδομένων στη μνήμη. Μπορούμε να το κάνουμε αυτό στην Python δημιουργώντας μια μεταβλητή που θα δείχνει σε ένα αντικείμενο. Πριν μπούμε στο πώς γίνεται αυτό, ας γνωρίσουμε πρώτα τα αντικείμενα και τις μεταβλητές.

1.3.2. Αντικείμενα

Όταν εισάγουμε ή δημιουργούμε δεδομένα στην Python, ουσιαστικά δημιουργούμε ένα αντικείμενο στη μνήμη με μια μεταβλητή. Αυτές οι δύο λέξεις, **αντικείμενο** και **μεταβλητή** σημαίνουν ελαφρώς διαφορετικά πράγματα, αλλά συχνά χρησιμοποιούνται εναλλακτικά. Δεν θα μπούμε στις περιπλοκές των διαφορών τους και γιατί υπάρχουν σε αυτό το εγχειρίδιο, αλλά προς το παρόν, δείτε ένα αντικείμενο ως κάτι που δημιουργείται από ένα σενάριο Python και αποθηκεύεται στη μνήμη του υπολογιστή σας ώστε να μπορεί να χρησιμοποιηθεί αργότερα σε ένα πρόγραμμα.

Σκεφτείτε τη **μνήμη του υπολογιστή σας** παρόμοια με τον εγκέφαλό σας. Φανταστείτε εάν χρειάστηκε να θυμάστε ποιο είναι το γερμανικό λάθος για “γεια σας”. Μπορεί να χρησιμοποιήσετε τη μνήμη σας παρόμοια με μια κάρτα φλας, όπου “γεια σας” στα αγγλικά ισοδυναμεί με “hallo” στα γερμανικά. Στην Python, δημιουργούμε αντικείμενα με παρόμοιο τρόπο. Το αντικείμενο θα ήταν η καταχώρηση λεξικού του “hello: hallo”.

1.3.3. Μεταβλητές

Προκειμένου να αναφέρετε αυτήν την καταχώρηση λεξικού στη μνήμη, πρέπει να έχουμε ένα τρόπο για να το αναφέρετε. Το κάνουμε αυτό με μια μεταβλητή. Η μεταβλητή είναι απλώς το όνομα του στοιχείου στο σενάριό μας που θα δείχνει στο αντικείμενο αυτό στη μνήμη. Οι μεταβλητές

κάνουν ώστε να μπορούμε να έχουμε ένα εύκολο να θυμηθούμε όνομα για να αναφέρουμε, κάλεσμα και αλλάξουμε αυτό το αντικείμενο στη μνήμη.

Οι μεταβλητές μπορούν να δημιουργηθούν πληκτρολογώντας μια μοναδική λέξη, ακολουθούμενη από ένα σημάδι =, ακολουθούμενο από τα συγκεκριμένα δεδομένα. Όπως θα μάθουμε σε όλο αυτό το κεφάλαιο, υπάρχουν πολλοί τύποι δεδομένων που δημιουργούνται διαφορετικά. Ας δημιουργήσουμε το πρώτο μας αντικείμενο πριν ξεκινήσουμε. Αυτό θα είναι μια συμβολοσειρά, ή ένα κομμάτι κειμένου. (Θα μάθουμε για αυτές με περισσότερες λεπτομέρειες παρακάτω.) Στη δική μου περίπτωση, θέλω να δημιουργήσω το αντικείμενο συγγραφέα. Θέλω ο συγγραφέας να σχετίζεται με το όνομά μου στη μνήμη. Στο κελί, ή μπλοκ κώδικα, παρακάτω, ας το κάνουμε αυτό.

```
author = "William Mattingly"
```

Εξαίρετα! Δημιουργήσαμε το πρώτο μας αντικείμενο. Τώρα, ήρθε η ώρα να χρησιμοποιήσουμε αυτό το αντικείμενο. Παρακάτω, θα μάθουμε για τρόπους που μπορούμε να χειριστούμε συμβολοσειρές, αλλά προς το παρόν, ας απλώς δούμε αν αυτό το αντικείμενο υπάρχει στη μνήμη. Μπορούμε να το κάνουμε αυτό με τη συνάρτηση print.

Η συνάρτηση print θα γίνει ο καλύτερος φίλος σας στην Python. Είναι, ίσως, η συνάρτηση που χρησιμοποιώ πιο συνήθως. Ο λόγος είναι ότι η συνάρτηση print σας επιτρέπει να αποσφαλμώσετε εύκολα, ή να αναγνωρίσετε προβλήματα και να τα διορθώσετε, μέσα στον κώδικά σας. Μας επιτρέπει να εκτυπώνουμε αντικείμενα που αποθηκεύονται στη μνήμη.

Για να χρησιμοποιήσετε τη συνάρτηση print, πληκτρολογούμε τη λέξη print ακολουθούμενη από μια ανοικτή παρενθέση. Μετά την ανοικτή παρενθέση, τοποθετούμε το αντικείμενο ή εκείνο το κομμάτι δεδομένων που θέλουμε να εκτυπώσουμε. Μετά από αυτό, κλείνουμε τη συνάρτηση με τη κλειστή παρενθέσεση. Ας δοκιμάσουμε να εκτυπώσουμε το νέο μας αντικείμενο συγγραφέα για να βεβαιωθούμε ότι είναι στη μνήμη.

```
print(author)
```

```
William Mattingly
```

Παρατηρήστε ότι όταν εκτελώ το κελί παραπάνω, βλέπω μια έξοδο που σχετίζεται με το αντικείμενο που δημιουργήσαμε παραπάνω. Τι θα συμβεί αν δοκιμάσω να εκτυπώσω αυτό το

αντικείμενο, αλλά χρησιμοποίησα ένα κεφαλαίο γράμμα, παρά ένα μικρό στην αρχή, έτσι Συγγραφέας, παρά συγγραφέας;

1.3.4. Ευαισθησία στα κεφαλαία

```
print(Author)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[22], line 1
----> 1 print(Author)

NameError: name 'Author' is not defined
```

Το τρομακτικό μπλοκ κειμένου παραπάνω δείχνει ότι έχουμε παράγει ένα σφάλμα στην Python. Αυτό το λάθος μας διδάσκει δύο πράγματα. Πρώτον, η Python είναι ευαίσθητη στην περίπτωση. Αυτό σημαίνει ότι εάν κάποιο αντικείμενο (ή συμβολοσειρά) θα χρειαστεί να ταιριάζει όχι μόνο τα γράμματα, αλλά και η περίπτωση αυτών των γραμμάτων. Δεύτερον, αυτό το λάθος μας διδάσκει ότι μπορούμε μόνο να καλέσουμε αντικείμενα που έχουν δημιουργηθεί και αποθηκευθεί στη μνήμη.

Τώρα που έχουμε τη μεταβλητή να δείχνει σε ένα συγκεκριμένο κομμάτι δεδομένων, μπορούμε να κάνουμε τη συνάρτηση print παραπάνω λίγο πιο δυναμική. Ας δοκιμάσουμε και να εκτυπώσουμε την ίδια δήλωση όπως πριν, αλλά με το νέο πλήρες όνομα συγγραφέα. Δεν περιμένω ότι θα καταλάβετε τις λεπτομέρειες του κώδικα παρακάτω, παρά απλώς κατανοήστε ότι συχνά θα χρειαστούμε να αποθηκεύσουμε μεταβλητές στη μνήμη ώστε να μπορούμε να τις χρησιμοποιήσουμε αργότερα στα προγράμματά μας.

```
print(f"Hello, {author}!")
```

```
Hello, William Mattingly!
```

1.3.5. Δεσμευμένες Λέξεις

Κατά την εργασία με την Python, υπάρχει ένας αριθμός λέξεων γνωστών ως **δεσμευμένες λέξεις**. Αυτές είναι λέξεις που δεν μπορούν να χρησιμοποιηθούν ως ονόματα μεταβλητών. Ως έκδοση

Python 3.6, υπάρχουν συνολικά 33 δεσμευμένες λέξεις. Σε μπορεί μερικές φορές να είναι δύσκολο να θυμάστε όλες αυτές τις δεσμευμένες λέξεις, οπότε η Python έχει μια ωραία ενσωματωμένη συνάρτηση, "help". Εάν εκτελέσουμε την ακόλουθη εντολή, θα δούμε ένα ολόκληρο κατάλογο.

```
help("keywords")
```

```
Here is a list of the Python keywords.  Enter any keyword to get more help.

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

Αυτές είναι λέξεις που δεν μπορείτε να χρησιμοποιήσετε ως όνομα μεταβλητής.

1.3.6. Ενσωματωμένοι Τύποι

Επιπλέον των δεσμευμένων λέξεων, υπάρχουν επίσης ενσωματωμένοι τύποι στην Python. Αυτές είναι λέξεις που μπορείτε να χρησιμοποιήσετε (όπως θα δούμε) για να μετατρέψετε έναν τύπο δεδομένων σε άλλον. Υπάρχουν συνολικά 94 από αυτές. Σε αντίθεση με τις δεσμευμένες λέξεις, μπορείτε * να χρησιμοποιήσετε ένα ενσωματωμένο τύπο ως όνομα μεταβλητής. Ωστόσο, είναι ισχυρά αποθαρρύνεται να το κάνετε γιατί θα αντικαταστήσει την προορισμένη χρήση αυτών των ονομάτων μεταβλητών στο σενάριό σας.

```
import builtins
[getattr(builtins, d) for d in dir(builtins) if isinstance(getattr(builtins, d), ty
```

```
[ArithmeticError,
 AssertionError,
 AttributeError,
 BaseException,
 BlockingIOError,
 BrokenPipeError,
 BufferError,
 BytesWarning,
 ChildProcessError,
 ConnectionAbortedError,
 ConnectionError,
 ConnectionRefusedError,
 ConnectionResetError,
 DeprecationWarning,
 EOFError,
 OSError,
 Exception,
 FileExistsError,
 FileNotFoundError,
 FloatingPointError,
 FutureWarning,
 GeneratorExit,
 OSError,
 ImportError,
 ImportWarning,
 IndentationError,
 IndexError,
 InterruptedError,
 IsADirectoryError,
 KeyError,
 KeyboardInterrupt,
 LookupError,
 MemoryError,
 ModuleNotFoundError,
 NameError,
 NotADirectoryError,
 NotImplementedError,
 OSError,
 OverflowError,
 PendingDeprecationWarning,
 PermissionError,
 ProcessLookupError,
 RecursionError,
 ReferenceError,
 ResourceWarning,
 RuntimeError,
 RuntimeWarning,
 StopAsyncIteration,
 StopIteration,
 SyntaxError,
 SyntaxWarning,
 SystemError,
 SystemExit,
```

```
  TabError,
  TimeoutError,
  TypeError,
  UnboundLocalError,
  UnicodeDecodeError,
  UnicodeEncodeError,
  UnicodeError,
  UnicodeTranslateError,
  UnicodeWarning,
  UserWarning,
  ValueError,
  Warning,
  OSError,
  ZeroDivisionError,
  _frozen_importlib.BuiltinImporter,
  bool,
  bytearray,
  bytes,
  classmethod,
  complex,
  dict,
  enumerate,
  filter,
  float,
  frozenset,
  int,
  list,
  map,
  memoryview,
  object,
  property,
  range,
  reversed,
  set,
  slice,
  staticmethod,
  str,
  super,
  tuple,
  type,
  zip]
```

Από αυτή τη μεγάλη λίστα, συνιστώ να δώσετε ιδιαίτερη προσοχή σε αυτούς που είναι πιο πιθανό να γράψετε φυσικά: bool, dict, float, int, list, map, object, property, range, reversed, set, slice, str, super, tuple, type, και zip. Είναι πιο πιθανό να χρησιμοποιήσετε αυτά ως ονόματα μεταβλητών κατά λάθος από, ας πούμε, ZeroDivisionError· και αυτή η μικρότερη λίστα είναι πολύ πιο εύκολη να απομνημονεύσετε.

1.3.7. Συνάρτηση Τύπου

Είναι συχνά απαραίτητο να ελέγξετε τι τύπο δεδομένων είναι μια μεταβλητή. Για να το αναγνωρίσετε, μπορείτε να χρησιμοποιήσετε τη ενσωματωμένη συνάρτηση **type** στην Python. Για να χρησιμοποιήσετε type, χρησιμοποιούμε την εντολή παρακάτω με τα δεδομένα που θέλουμε να αναλύσουμε τοποθετημένα μεταξύ των δύο παρενθέσεων.

```
type("this is a string...")
```

```
str
```

1.3.8. Σφάλματα

Σε όλη τη προγραμματιστική σας καριέρα, συχνά θα διαβάζετε ή θα ακούσετε για `bugs` (σφάλματα). Ένα bug είναι ένα πρόβλημα στον κώδικά σας που είτε επιστρέφει ένα σφάλμα είτε ένα απροσδόκητο αποτέλεσμα στην έξοδο. Η ανίχνευση των bugs και η διόρθωσή τους είναι γνωστή ως `debugging` (αποσφαλμάτωση). Εκτός από το να μάθετε πώς να κωδικοποιήσετε λύσεις σε προβλήματα, `debugging` μπορεί να είναι ένας από τους πιο χρονοβόρους πτυχές της συγγραφής ενός προγράμματος, ειδικά αν είναι αρκετά σύνθετο. Σε όλο αυτό το εγχειρίδιο, θα συναντήσουμε συνήθη σφάλματα ώστε να μπορείτε να τα δείτε σε αυτό τον ελεγχόμενο χώρο. Θα περιηγηθούμε επίσης τι σημαίνει το σφάλμα και πώς να το διορθώσετε. Ωστόσο, είναι πιθανό να δημιουργήσετε πολλά άλλα bugs καθώς δοκιμάζετε να εφαρμόσετε τον κώδικα σε αυτό το εγχειρίδιο στα δικά σας δεδομένα. Αυτό αναμένεται. Θυμηθείτε πάντα να διαβάσετε προσεκτικά κάθε γραμμή του κώδικα Python σας και, εάν έχετε ένα σφάλμα, αναγνωρίστε από πού προέρχεται το σφάλμα και ποιο είναι.

2. Δεδομένα και Δομές Δεδομένων

Θέματα που Καλύπτονται σε αυτό το Κεφάλαιο

1. Τι είναι Δεδομένα;
2. Τύποι Δεδομένων της Python
3. Συμβολοσειρές
4. Αριθμοί (Ακέραιοι και Δεκαδικοί)

5. Λογικές Τιμές
6. Χειρισμός Συμβολοσειρών
7. Μαθηματικές Πράξεις
8. Τι είναι Δομή Δεδομένων;
9. Λίστες
10. Πλειάδες
11. Σύνολα
12. Λεξικά
13. Ευρετηρίαση

Σε αυτό το κεφάλαιο θα μάθουμε για δεδομένα και δομές δεδομένων. Όπως και σε κάθε γλώσσα προγραμματισμού, είναι σημαντικό να έχουμε βασική κατανόηση αυτών των δύο εννοιών καθώς είναι τα δομικά στοιχεία για τα περισσότερα πράγματα που θα κάνετε στην Python. Κάθε φορά που γράφετε ένα πρόγραμμα ή κάνετε κάτι με κώδικα, ουσιαστικά γράφετε εντολές για φόρτωση, αποθήκευση, αλληλεπίδραση και χειρισμό δεδομένων με κάποιο τρόπο. Η κατανόηση των διαφορετικών τύπων δεδομένων και του τρόπου δόμησης και αποθήκευσής τους είναι, επομένως, απαραίτητη.

2.1. Εισαγωγή στα Δεδομένα

2.1.1. Τι είναι τα Δεδομένα;

Στην Python υπάρχουν επτά βασικά στοιχεία δεδομένων και δομές δεδομένων με τα οποία θα δουλεύουμε:

- συμβολοσειρές (κείμενο)
- ακέραιοι (ακέραιοι αριθμοί)
- δεκαδικοί (δεκαδικοί αριθμοί)
- Boolean (Αληθής ή Ψευδής)
- λίστες
- πλειάδες (λίστες που δεν μπορούν να αλλάξουν στη μνήμη)
- λεξικά (κλειδί : τιμή)

Σε αυτό το κεφάλαιο, θα εξερευνήσουμε καθένα από αυτά. Ενώ αυτή η ενότητα εστιάζει στα δεδομένα: συμβολοσειρές, ακέραιοι, δεκαδικοί και Boolean· η επόμενη ενότητα θα εστιάσει στις δομές δεδομένων: λίστες, πλειάδες και λεξικά.

Τα δεδομένα είναι τμήματα πληροφοριών (ο ενικός είναι datum)· για παράδειγμα, ακέραιοι, δεκαδικοί και συμβολοσειρές. Οι δομές δεδομένων είναι αντικείμενα που καθιστούν τα δεδομένα σχεσιακά, δηλαδή λίστες, πλειάδες και λεξικά. Ξεκινήστε να εκπαιδεύετε τον εγκέφαλό σας να αναγνωρίζει τη σύνταξη της Python για αυτά τα στοιχεία δεδομένων και τις δομές δεδομένων που συζητούνται παρακάτω.

Για τη διευκόλυνσή σας, ακολουθεί ένας πίνακας που σας βοηθά να ανατρέξετε σε αυτούς τους τύπους δεδομένων.

Πίνακας για Τύπους Δεδομένων και Δομές Δεδομένων στην Python

Όνομα	Τύπος	Κατηγορία	Παράδειγμα	Τέχνασμα	Μεταβλητό
συμβολοσειρά	`str`	κείμενο	`"William"`	εισαγωγικά	όχι
ακέραιος	`int`	αριθμός	`1`	χωρίς δεκαδικό	όχι
δεκαδικός	`float`	αριθμός	`1.1`	με δεκαδικό	όχι
Boolean	`bool`	boolean	`True`	Αληθής ή Ψευδής	όχι
λίστα	`list`	ακολουθία	`[1, 1.1, "one"]`	[]	ναι
πλειάδα	`tuple`	ακολουθία	`(1, 1.1, "one")`	()	όχι
σύνολο	`set`	ακολουθία	`{1, 1.1, "one"}`	{} - μοναδικό	ναι
λεξικό	`dict`	χαρτογραφία	`{"name": "tom"}`	{κλειδί:τιμή}	ναι

2.1.2. Συμβολοσειρές

Οι συμβολοσειρές είναι μια ακολουθία χαρακτήρων. Ένας καλός τρόπος να σκεφτείτε μια συμβολοσειρά είναι ως κείμενο. Μπορούμε να δημιουργήσουμε μια συμβολοσειρά στην Python καθορίζοντας ένα μοναδικό όνομα μεταβλητής και ορίζοντας = σε κάτι μέσα σε εισαγωγικά, δηλαδή " " ή ' '.

Το άνοιγμα ενός εισαγωγικού υποδεικνύει στην Python ότι έχει ξεκινήσει μια συμβολοσειρά και το κλείσιμο του ίδιου στυλ εισαγωγικού υποδεικνύει το κλείσιμο της συμβολοσειράς. Είναι σημαντικό να χρησιμοποιείτε το ίδιο στυλ εισαγωγικού για μια συμβολοσειρά, είτε διπλό είτε μονό.

Ας δούμε μερικά παραδείγματα συμβολοσειρών.

2.1.2.1. Παραδείγματα Συμβολοσειρών

Στο πρώτο μας παράδειγμα μιας συμβολοσειράς, θα ονομάσουμε το πρώτο μας αντικείμενο συμβολοσειράς με το όνομα μεταβλητής `first_string`. Παρατηρήστε ότι χρησιμοποιούμε υπογράμμιση (`_`) για να υποδείξουμε το διαχωρισμό δύο λέξεων στο όνομα της μεταβλητής μας. Αυτή είναι κοινή πρακτική στην Python. Δεν μπορούμε να χρησιμοποιήσουμε ένα κενό σε ένα όνομα μεταβλητής. Μια άλλη σύμβαση ονοματολογίας θα μπορούσε να ήταν `firstString`, όπου η πρώτη λέξη είναι πεζή ενώ η αρχή κάθε επακόλουθης λέξης κεφαλαιοποιημένη. Αυτή η σύμβαση ονοματολογίας, αν και αποδεκτή, είναι πιο συνηθισμένη σε άλλες γλώσσες προγραμματισμού, όπως η JavaScript.

```
first_string = "This is a string."
```

Τώρα που έχουμε δημιουργήσει το πρώτο μας αντικείμενο, ας προσπαθήσουμε να το εκτυπώσουμε. Μπορούμε να το κάνουμε αυτό χρησιμοποιώντας τη συνάρτηση `print`.

```
print(first_string)
```

```
This is a string.
```

Ας δούμε ένα άλλο παράδειγμα. Αυτή τη φορά, ωστόσο, θα χρησιμοποιήσουμε δύο `'`.

```
second_string = 'This is a string too.'
print(second_string)
```

```
This is a string too.
```

Στην εφαρμογή Trinket παρακάτω, δημιουργήστε μια συμβολοσειρά και εκτυπώστε την.

```
from IPython.display import IFrame
IFrame('https://trinket.io/embed/python3/3fe4c8f3f4', 700, 500)
```

Ας δούμε τώρα ένα κακό παράδειγμα μιας συμβολοσειράς. Στο παράδειγμα παρακάτω, θα προσπαθήσουμε σκόπιμα να δημιουργήσουμε ένα μήνυμα σφάλματος συνδυάζοντας δύο διαφορετικούς τύπους εισαγωγικών.

```
bad_string = "This is a bad example of a string'
```

```
  Input In [6]
    bad_string = "This is a bad example of a string'
                                                    ^
SyntaxError: EOL while scanning string literal
```

Αυτό το μήνυμα σφάλματος μας ενημερώνει ότι έχουμε ενεργοποιήσει ένα “SyntaxError”. Ένα SyntaxError είναι ένα σφάλμα Python στο οποίο έχουμε χρησιμοποιήσει ακατάλληλη σύνταξη, ή γλώσσα κωδικοποίησης. Αυτό σημαίνει ότι όταν ο υπολογιστής μας προσπαθεί να εκτελέσει τον παραπάνω κώδικα, δεν ξέρει πώς να ερμηνεύσει τη γραμμή. Θα συναντήσετε συχνά σφάλματα σαν αυτό στο ταξίδι σας ως προγραμματιστής.

Είναι πάντα σημαντικό να διαβάσετε το μήνυμα σφάλματος καθώς θα σας ενημερώσει για τις ιδιαιτερότητες του προβλήματος. Για παράδειγμα, μπορώ να δω ότι το σφάλμα μου ενεργοποιήθηκε λόγω κάτι στο κελί εισόδου **6**. Εάν εκτελούσατε αυτήν την ίδια εντολή εντός ενός σεναρίου Python, θα δείτε τη συγκεκριμένη γραμμή που ενεργοποίησε το σφάλμα. Αυτό σας επιτρέπει να ξεκινήσετε να **αποσφαλματώσετε** ή να προσδιορίσετε την πηγή του σφάλματος και να το διορθώσετε. Εάν, για κάποιο λόγο, δεν μπορείτε να κατανοήσετε την αιτία του σφάλματος, ίσως είναι χρήσιμο να το αναζητήσετε στο Google και να ψάξετε για απαντήσεις σε φόρουμ, όπως το StackOverflow.

Μερικές φορές, θα είναι απαραίτητο να έχετε μια πολύ μακρά συμβολοσειρά που εκτείνεται σε πολλές γραμμές εντός ενός σεναρίου Python. Σε αυτές τις περιπτώσεις, μπορείτε να χρησιμοποιήσετε τρία του ίδιου στυλ εισαγωγικού (μονό ή διπλό) συνεπώς για να δημιουργήσετε μια συμβολοσειρά σε πολλές γραμμές.

```
long_string = '''

This is a verrry long string.

'''
```

```
print(long_string)
```

```
This is a verrry long string.
```

2.1.3. Εργασία με Συμβολοσειρές ως Δεδομένα

Συχνά όταν εργάζεστε εντός της Python, δεν θα απλώς δημιουργείτε δεδομένα, θα τα χειρίζεστε και θα τα αλλάζετε. Επειδή οι kubernητές συχνά εργάζονται με συμβολοσειρές, σας συνιστώ να ξοδεύσετε ένα καλό χρονικό διάστημα στην εξάσκηση και στη σύγχυση των βασικών τρόπων με τους οποίους εργαζόμαστε με συμβολοσειρές στην Python μέσω των ενσωματωμένων μεθόδων.

Για να αλληλεπιδράσετε με τις συμβολοσειρές ως τμήματα δεδομένων, χρησιμοποιούμε **μεθόδους** και **συναρτήσεις**. Οι κύριες συναρτήσεις για την αλληλεπίδραση με συμβολοσειρές σε βασικό επίπεδο είναι ενσωματωμένες στην Python. Αυτό σημαίνει ότι δεν χρειάζεται να εγκαταστήσετε βιβλιοθήκες τρίτων μερών. Αργότερα σε αυτό το σχολικό βιβλίο θα κάνουμε πιο προχωρημένα πράγματα με συμβολοσειρές χρησιμοποιώντας βιβλιοθήκες τρίτων μερών, όπως η Regex, αλλά για τώρα, θα δουλεύουμε απλώς με τις βασικές μεθόδους.

Ας μάθουμε να χειριζόμαστε τις συμβολοσειρές τώρα μέσω κώδικα, αλλά πρώτα πρέπει να δημιουργήσουμε μια συμβολοσειρά. Ας την καλέσουμε πρόταση.

```
sentence = "I am going to learn how to program in Python!"
```

2.1.3.1. Μέθοδος Upper

Δεν είναι πολύ έξυπνο όνομα, αλλά θα λειτουργήσει για τους σκοπούς μας. Τώρα, ας προσπαθήσουμε να μετατρέψουμε ολόκληρη τη συμβολοσειρά σε κεφαλαία. Μπορούμε να το κάνουμε αυτό με τη μέθοδο `.upper()`. Παρατηρήστε ότι το `.upper()` έρχεται μετά τη συμβολοσειρά και εντός του `()` δεν υπάρχουν επιχειρήματα. Αυτός είναι ένας τρόπος με τον οποίο μπορείτε εύκολα να αναγνωρίσετε μια μέθοδο (σε αντίθεση με μια συνάρτηση). Θα μάθουμε περισσότερα για αυτές τις διακρίσεις στα κεφάλαια για τις συναρτήσεις και τις κλάσεις.

```
print(sentence.upper())
```

```
I AM GOING TO LEARN HOW TO PROGRAM IN PYTHON!
```

2.1.3.2. Μέθοδος Lower

Παρατηρήστε ότι η συμβολοσειρά μας είναι τώρα όλα κεφαλαία. Μπορούμε να κάνουμε το ίδιο με τη μέθοδο `.lower()`, αλλά αυτή η μέθοδος θα κάνει τα πάντα στη συμβολοσειρά πεζά.

```
print(sentence.lower())
```

```
i am going to learn how to program in python!
```

2.1.3.3. Μέθοδος Capitalize

Στην επιφάνεια, αυτές οι μέθοδοι ενδέχεται να φαίνεται ότι είναι χρήσιμες μόνο σε ειδικές περιπτώσεις. Ενώ αυτές οι μέθοδοι είναι χρήσιμες για να κάνουν τις συμβολοσειρές να φαίνονται όπως θέλετε, έχουν πολύ μεγαλύτερη χρησιμότητα. Φανταστείτε ότι θέλατε να αναζητήσετε ένα όνομα, “William”, σε μια συμβολοσειρά. Τι γίνεται αν τα δεδομένα που εξετάζετε προέρχονται από μηνύματα ηλεκτρονικού ταχυδρομείου, μηνύματα κειμένου κ.λπ.; Το William ενδέχεται να είναι κεφαλαιοποιημένο ή όχι. Αυτό σημαίνει ότι θα έπρεπε να εκτελέσετε δύο αναζητήσεις για τον William σε μια συμβολοσειρά. Εάν, ωστόσο, μετατρέψετε τη συμβολοσειρά σε πεζά πριν αναζητήσετε, μπορείτε απλώς να αναζητήσετε “william” και θα βρείτε όλα τα ταιριάσματα. Αυτό είναι ένα από τα πράγματα που συμβαίνουν στο back-end των περισσότερων μηχανών αναζήτησης για να εξασφαλίσουν ότι η αναζήτησή σας δεν είναι αυστηρά ευαίσθητη στην περίπτωση. Στην

Python, ωστόσο, είναι σημαντικό να πραγματοποιήσετε αυτό το βήμα καθαρισμού δεδομένων πριν εκτελέσετε αναζητήσεις πάνω από συμβολοσειρές.

Ας εξερευνήσουμε μια άλλη μέθοδο, `.capitalize()`. Αυτή η μέθοδος θα σας επιτρέψει να κεφαλαιοποιήσετε μια συμβολοσειρά.

```
first_name = "william"
```

```
print(first_name.capitalize())
```

```
William
```

Θα χρησιμοποιήσω αυτό σε ειδικές περιπτώσεις, ιδιαίτερα όταν πραγματοποιώ καθαρισμό δεδομένων και πρέπει να εξασφαλίσω ότι όλα τα ονόματα ή τα κύρια ονόματα σε ένα σύνολο δεδομένων είναι καθαρά και καλά δομημένα.

2.1.3.4. Μέθοδος Replace

Ίσως η πιο χρήσιμη μέθοδος συμβολοσειράς είναι η .replace(). Παρατηρήστε στα κελιά παρακάτω, η αντικατάσταση παίρνει υποχρεωτικά δύο ορίσματα, ή πράγματα που περνάνε μεταξύ των παρενθέσεων. Το καθένα χωρίζεται με κόμμα. Το πρώτο όρισμα είναι η υποσυμβολοσειρά ή μέρος της συμβολοσειράς που θέλετε να αντικαταστήσετε και το δεύτερο όρισμα είναι αυτό με το οποίο θέλετε να την αντικαταστήσετε. Γιατί είναι αυτό τόσο χρήσιμο; Εάν χρησιμοποιείτε την Python για την ανάλυση κειμένων, εκείνα τα κείμενα θα, σας υπόσχομαι, δεν θα είναι ποτέ καλά καθαρά. Μπορεί να έχουν κακή κωδικοποίηση, χαρακτήρες που θα ρίξουν τις αναζητήσεις, κακή OCR, πολλαπλά διαλείμματα γραμμών, χαρακτήρες που έχουν αποσπασθεί με διακεκομμένη, η λίστα συνεχίζεται. Η `.replace()` μέθοδος σας επιτρέπει να καθαρίσετε γρήγορα και αποτελεσματικά τα κειμενικά δεδομένα ώστε να μπορούν να τυποποιηθούν.

Σε αντίθεση με τις παραπάνω μεθόδους, για `.replace()`, πρέπει να βάλουμε δύο πράγματα εντός των παρενθέσεων. Αυτές είναι γνωστές ως επιχειρήματα. Αυτή η μέθοδος απαιτεί δύο και πρέπει να είναι σε σειρά. Το πρώτο είναι το πράγμα που θέλετε να αντικαταστήσετε και το δεύτερο είναι το πράγμα που θέλετε να την αντικαταστήσετε. Αυτές θα χωριστούν με παρένθεση και και οι δύο πρέπει να είναι συμβολοσειρές. Θα πρέπει, επομένως, να μοιάζει κάτι σαν αυτό:

```
.replace("το πράγμα προς αντικατάσταση", "το πράγμα που θέλετε να κάνετε δεν το αντ
```

Στο παράδειγμα παρακάτω, ας προσπαθήσουμε να αντικαταστήσουμε την τελεία στο τέλος του "Mattingly."

```
introduction = "My name is William Mattingly."
```

```
print(introduction.replace(".", ""))
```

```
My name is William Mattingly
```

Εξαιρετικά! Τώρα, ας προσπαθήσουμε να ξανατυπώσουμε `introduction_sentence` και να δούμε τι συμβαίνει.

```
print(introduction)
```

```
My name is William Mattingly.
```

Ω όχι! Κάτι δεν είναι σωστό. Δεν έχει αλλάξει τίποτα! Πράγματι, αυτό συμβαίνει επειδή οι συμβολοσειρές είναι **αμετάβλητα αντικείμενα**. Τα αμετάβλητα αντικείμενα είναι αντικείμενα που δεν μπορούν να αλλάξουν στη μνήμη. Όπως θα δούμε στο επόμενο κεφάλαιο με λίστες, ο άλλος τύπος αντικειμένου είναι ένα που *μπορεί* να αλλάξει στη μνήμη. Αυτές είναι γνωστές ως **μεταβλητά αντικείμενα**. Για να αλλάξετε μια συμβολοσειρά, επομένως, πρέπει να την ξαναδημιουργήσετε στη μνήμη ή να δημιουργήσετε ένα νέο αντικείμενο συμβολοσειράς από αυτό. Ας προσπαθήσουμε να κάνουμε αυτό παρακάτω.

```
new_introduction = introduction.replace(".", "")
```

```
print(new_introduction)
```

```
My name is William Mattingly
```

2.1.3.5. Μέθοδος Split

Οι συμβολοσειρές έχουν πολλές άλλες χρήσιμες μεθόδους που θα μάθουμε σε ολόκληρο αυτό το σχολικό βιβλίο, όπως η μέθοδος `split()` που επιστρέφει μια λίστα υποσυμβολοσειρών ή μικρότερων συμβολοσειρών, που χωρίζονται από το διαχωριστικό, το οποίο είναι το όρισμα της μεθόδου. Το διαχωριστικό λέει στην Python πώς να χωρίσει τη συμβολοσειρά. Από προεπιλογή, η `split()` θα χωρίσει τη συμβολοσειρά σας στο κενό. Ας προσπαθήσουμε να χωρίσουμε την ακόλουθη συμβολοσειρά.

```
book_name = "Harry Potter and the Chamber of Secrets"
```

Όπως με το Replace, το Split είναι μια μέθοδος και, επομένως, το χρησιμοποιούμε με μια τελεία μετά το όνομα της μεταβλητής.

```
print(book_name.split())
```

```
['Harry', 'Potter', 'and', 'the', 'Chamber', 'of', 'Secrets']
```

Το όνομα του βιβλίου μας χωρίζεται τώρα σε μεμονωμένες λέξεις χάρη στο διαχωριστικό προεπιλογής της συνάρτησης split, το οποίο είναι το κενό. Παρατηρήστε, ωστόσο, ότι το διαχωριστικό εξαφανίζεται από τη λίστα μας των υποσυμβολοσειρών. Στο επόμενο κεφάλαιο, θα μάθουμε για τις δομές δεδομένων, όπως τις λίστες. Σε εκείνη την ενότητα, θα μάθουμε πώς να πάρουμε συγκεκριμένα ευρετήρια, ή ενότητες, της λίστας.

Το Split μπορεί επίσης να λάβει ένα όρισμα που προσδιορίζει πού να χωρίσει μια συμβολοσειρά, για παράδειγμα, κάτι άλλο από ένα κενό. Ας υποθέσουμε ότι ήμουν ενδιαferomenemoi στο να πάρω τον υπότιτλο του ονόματος του βιβλίου. Θα μπορούσα να χωρίσω τη συμβολοσειρά στο “ και “.

```
print(book_name.split(" and "))
```

```
['Harry Potter', 'the Chamber of Secrets']
```

Όπως μπορούμε να δούμε, έχουμε χωρίσει με επιτυχία τον τίτλο από τον υπότιτλο. Θα δουλεύουμε με συμβολοσειρές πολλές φορές περισσότερο καθώς προχωράμε σε αυτό το σχολικό βιβλίο. Θα πρέπει σε αυτό το σημείο να είστε εξοικειωμένοι με το τι είναι οι συμβολοσειρές, πώς να τις

δημιουργήσετε και πώς να αλληλεπιδράσετε με αυτές μέσω ορισμένων βασικών μεθόδων, όπως upper(), lower(), capitalize(), replace() και split().

2.1.4. Αριθμοί (Ακέραιοι και Δεκαδικοί)

Οι αριθμοί αναπαρίστανται σε γλώσσες προγραμματισμού με διάφορους τρόπους. Οι δύο που θα αντιμετωπίσουμε είναι ακέραιοι και δεκαδικοί.

Ένας **ακέραιος** είναι ένα ψηφίο που δεν περιέχει δεκαδικό σημείο, δηλαδή 1 ή 2 ή 3. Αυτό μπορεί να είναι αριθμός οποιουδήποτε μεγέθους, όπως 100.001.200. Ένας δεκαδικός, από την άλλη πλευρά, είναι ένα ψηφίο με δεκαδικό σημείο. Έτσι, ενώ 1 είναι ακέραιος, 1,0 είναι δεκαδικός. Οι δεκαδικοί, όπως και οι ακέραιοι, μπορούν να είναι οποιουδήποτε μεγέθους, αλλά αναγκαστικά έχουν δεκαδικό σημείο, δηλαδή 200.0020002938. Στην Python δεν χρειάζεστε ειδικούς χαρακτήρες για να δημιουργήσετε ένα ακέραιο ή δεκαδικό αντικείμενο. Χρειάζεστε απλώς ένα σύμβολο ίσου. Στο παράδειγμα παρακάτω, έχουμε δύο αντικείμενα που δημιουργούνται με ένα μόνο σύμβολο ίσου. Αυτά τα αντικείμενα ονομάζονται an_integer και a_float με το πρώτο να είναι ένα αντικείμενο που αντιστοιχεί στον ακέραιο 1 και το δεύτερο να είναι ένα αντικείμενο που αντιστοιχεί στο δεκαδικό 1,1.

2.1.4.1. Παραδείγματα Αριθμών

```
int1 = 1
```

```
print(int1)
```

```
1
```

```
float1 = 1.1
```

```
print (float1)
```

```
1.1
```

2.1.5. Εργασία με Αριθμούς ως Δεδομένα

Τώρα που καταλαβαίνετε πώς λειτουργούν οι συμβολοσειρές, ας ξεκινήσουμε να εξερευνούμε έναν άλλο τύπο δεδομένων: τους αριθμούς. Οι αριθμοί στην Python υπάρχουν σε δύο κύρια είδη:

- ακέραιοι
- δεκαδικοί

Όπως σημειώθηκε παραπάνω, οι ακέραιοι είναι αριθμοί χωρίς δεκαδικό σημείο, ενώ οι δεκαδικοί είναι αριθμοί με δεκαδικό σημείο. Αυτή είναι μια σημαντική διάκριση που θα πρέπει να θυμάστε, ιδιαίτερα όταν εργάζεστε με δεδομένα που εισάγονται και εξάγονται σε Excel.

Ως ερευνητές στις ψηφιακές ανθρωπιστικές επιστήμες, ίσως να σκέφτεστε στον εαυτό σας: "Εγώ απλώς δουλεύω με κείμενο, γιατί θα πρέπει να ενδιαφέρομαι τόσο πολύ για τους αριθμούς;" Η απάντηση; Οι αριθμοί μας επιτρέπουν να πραγματοποιούμε ποσοτική ανάλυση. Τι γίνεται αν θέλατε να γνωρίζετε πόσες φορές ένας συγκεκριμένος συγγραφέας έγραψε σε έναν συνάδελφο ή σε ποια σημεία έγραψε πιο συχνά, όπως συνέβη με το έργο της Δημοκρατίας των Γραμμάτων στο Πανεπιστήμιο Στάνφορντ;

Για να πραγματοποιήσετε αυτόν τον τύπο ανάλυσης, χρειάζεστε έλεγχο του τρόπου λειτουργίας των αριθμών στην Python, του τρόπου εκτέλεσης βασικών μαθηματικών συναρτήσεων σε αυτούς τους αριθμούς και του τρόπου αλληλεπίδρασης με αυτούς. Περαιτέρω, οι αριθμοί είναι απαραίτητοι για την κατανόηση για την εκτέλεση πιο προχωρημένων συναρτήσεων στην Python, όπως τα Loops, που διερευνώνται στο Κεφάλαιο 3.

Σε όλο το έργο σας στις ψηφιακές ανθρωπιστικές επιστήμες, είναι πολύ πιθανό ότι θα χρειαστεί να χειριστείτε αριθμούς μέσω μαθηματικών πράξεων. Ακολουθεί ένας πίνακας των κοινών πράξεων:

Πίνακας για Μαθηματικές Πράξεις

λειτουργία	κώδικας	περιγραφή	παράδειγμα	αποτέλεσμα
πρόσθεση	`+`	προσθέτει αριθμούς μαζί	`1+1`	2
αφαίρεση	`-`	αφαιρεί έναν αριθμό από άλλο	`1-1`	0
πολλαπλασιασμός	`*`	πολλαπλασιάζει δύο αριθμούς	`1*2`	2
εκθετική πολλαπλασιασμός	`**`	πραγματοποιεί εκθετική πολλαπλασιασμό	`2**2`	4
διαίρεση	`/`	χωρίζει έναν αριθμό από άλλο	`2/2`	1
modulo	`%`	υπόλοιπο	`2%7`	1
floor	`//`	ακέραιο πηλίκο	`2//7`	0

Ο τρόπος με τον οποίο δημιουργείτε ένα αντικείμενο αριθμού στην Python είναι να δημιουργήσετε ένα όνομα αντικειμένου, να χρησιμοποιήσετε το σύμβολο ίσου και να πληκτρολογήσετε τον αριθμό. Εάν ο αριθμός σας έχει ένα δεκαδικό, η Python θα τον θεωρήσει αυτόματα δεκαδικό. Εάν δεν έχει, θα τον θεωρήσει αυτόματα ακέραιο. Ας δημιουργήσουμε έναν ακέραιο και έναν δεκαδικό.

```
an_integer = 1
print(an_integer)
```

```
1
```

```
a_float = 1.1
print(a_float)
```

```
1.1
```

Στην εφαρμογή Trinket παρακάτω, προσπαθήστε να δημιουργήσετε έναν δεκαδικό και έναν ακέραιο και εκτυπώστε το καθένα.

```
IFrame('https://trinket.io/embed/python3/3fe4c8f3f4', 700, 500)
```

Τώρα που καταλαβαίνετε πώς να δημιουργήσετε έναν αριθμό στη μνήμη, ας προσπαθήσουμε να προσθέσουμε δύο αριθμούς μαζί. Στο παράδειγμα παρακάτω, θα προσθέσουμε 1 και 1 μαζί.

```
print(1+1)
```

```
2
```

Στην εφαρμογή Trinket παραπάνω, προσπαθήστε να εκτελέσετε τις άλλες μαθηματικές πράξεις.

2.1.6. Boolean

Ο όρος boolean προέρχεται από τη Δυαδική άλγεβρα, που είναι ένας τύπος μαθηματικών που λειτουργεί σε δυαδική λογική. Το δυαδικό είναι η βάση για όλους τους υπολογιστές, με εξαίρεση τους πιο νέους κβαντικούς υπολογιστές. Το δυαδικό είναι 0 ή 1· απενεργοποιημένο ή ενεργοποιημένο· αληθές ή ψευδές. Ένα αντικείμενο boolean σε γλώσσες προγραμματισμού είναι είτε `True` είτε `False`. Αληθής είναι 1, ενώ Ψευδής είναι 0. Στην Python μπορούμε να εκφράσουμε αυτές τις έννοιες με κεφαλαιοποιημένο T ή F σε True ή False. Ας δημιουργήσουμε τώρα ένα τέτοιο αντικείμενο.

2.1.6.1. Παραδείγματα Boolean

```
bool1 = True
```

```
print(bool1)
```

```
True
```

2.1.7. Συμπέρασμα

Αυτό το κεφάλαιο σας έχει εισαγάγει σε μερικούς από τους απαραίτητους τύπους δεδομένων: συμβολοσειρές, ακέραιοι, δεκαδικοί και boolean. Έχει επίσης εισαγάγει σας σε μερικές από τις βασικές μεθόδους και πράξεις που μπορείτε να εκτελέσετε σε συμβολοσειρές και αριθμούς. Πριν μεταβείτε στο επόμενο κεφάλαιο, συνιστώ να ξοδεύσετε λίγο χρόνο και δοκιμάστε αυτές τις μεθόδους στα δικά σας δεδομένα. Δοκιμάστε και χειριστείτε κείμενο εισόδου για να εντοπίσετε και να ανακτήσετε συγκεκριμένες πληροφορίες.

2.1.8. Ερωτήσεις Προκλήσεων

Ποια θα ήταν η έξοδος της παρακάτω εντολής;

```
type('What type of data is this?')
```

```
str
```

Τι λέτε γι' αυτό;

```
print("Hello, world!')
```

```
  Input In [19]
    print("Hello, world!')
                          ^
SyntaxError: EOL while scanning string literal
```

Και τι γι' αυτό;

```
person = "John"
person = person.lower()
person2 = person
person = person.upper()
print(person2)
```

▶ **Show code cell output**

Επεκτείνετε παρακάτω για να δείτε γιατί η έξοδος φαίνεται έτσι.

Εάν δεν το καταλάβατε σωστά, ας μιλήσουμε για το λόγο. Θυμηθείτε, ο προγραμματισμός είναι ένα ακολουθιακό σύνολο πράξεων.

Αριθμός	Γραμμή	Νόημα
1	person = “John”	καθορίζει ότι η μεταβλητή είναι μια συμβολοσειρά και ότι είναι “John”
2	person = person.lower()	αυτό θα καταστείλει τη συμβολοσειρά ώστε σε αυτό το στάδιο, θα ήταν “john”
3	person2 = person	αυτό δημιουργεί person2 ως μεταβλητή και το δείχνει στο person
4	person = person.upper()	αυτό αλλάζει τη μεταβλητή person και την κεφαλαιοποιεί. Η μεταβλητή person2, ωστόσο, παραμένει ίδια

2.1.9. Κουίζ

How do we create strings?

- With <>
- With ''
- With ""
- With :

What are the two types of numbers?

- Floats
- Booleans
- Integers
- Strings

What type of data is associated with text?

- Strings
- Floats
- Integers
- Boolean

What are the types of Booleans?

- True
- False
- true
- false

2.2. Εισαγωγή στις Δομές Δεδομένων

2.2.1. Δομές Δεδομένων

Στο τελευταίο κεφάλαιο, γνωρίσαμε συμβολοσειρές (strings), ακέραιους αριθμούς (integers), δεκαδικούς αριθμούς (floats) και τιμές Boolean (booleans). Καθεμία από αυτές ήταν ένας τύπος δεδομένων. Οι συμβολοσειρές, για παράδειγμα, μας επέτρεψαν να εργαζόμαστε με κείμενο και οι αριθμοί μας επέτρεψαν να εργαζόμαστε με ακέραιους και δεκαδικούς αριθμούς. Σε αυτό το κεφάλαιο, θα ξεκινήσουμε να εργαζόμαστε με δομές δεδομένων. **Δομές δεδομένων (Data structures)** είναι τρόποι αποθήκευσης πολλών ειδών δεδομένων με συστηματικό τρόπο. Στην Python, αυτές δημιουργούνται ως αντικείμενα που μπορούν να αποθηκευθούν στη μνήμη και να κληθούν αργότερα σε ένα σενάριο. Χωρίζονται σε δύο κατηγορίες: μεταβαλλόμενες (mutable) και αμετάβλητες (immutable). Συναντήσαμε αυτούς τους όρους στο τελευταίο κεφάλαιο, αλλά θα εξερευνήσουμε τι σημαίνουν με μεγαλύτερο βάθος παρακάτω.

Καθ' όλη την ενότητα, θα μάθουμε για ορισμένους από τους βασικούς τύπους δομών δεδομένων, πώς διαφέρουν και πώς μπορούν να χρησιμοποιηθούν. Θα καλύψουμε αυτές με επιχειρησιακό τρόπο. Σε όλο αυτό το εγχειρίδιο, θα χρησιμοποιούμε αυτές τις δομές δεδομένων καθώς γράφουμε κώδικα και εκτελούμε εργασίες καθαρισμού και ανάλυσης δεδομένων. Για να διατηρήσουμε τα πράγματα απλά προς το παρόν, θα επικεντρωθούμε σε τέσσερις τύπους δομών δεδομένων: λίστες (lists), πλειάδες (tuples), σύνολα (sets) και λεξικά (dictionaries). Υπάρχουν άλλοι τύποι δομών δεδομένων στην Python, αλλά αυτές είναι οι τέσσερις βασικές που θα χρησιμοποιήσετε πιο συχνά.

2.2.2. Λίστες

Η πρώτη δομή δεδομένων με την οποία θα εργαστούμε είναι γνωστή ως **λίστα (list)**. Οι λίστες είναι ακριβώς αυτό που ακούγεται, μια λίστα δεδομένων. Όπως θα δούμε παρακάτω, υπάρχουν πολλοί τρόποι αποθήκευσης πληροφοριών με τρόπο παρόμοιο με λίστες στην Python, όπως με πλειάδες (tuples) και σύνολα (sets), αλλά ο τρόπος που δημιουργούμε λίστες και ο τρόπος που αλληλεπιδρούμε με τις λίστες είναι διαφορετικός.

Οι λίστες και οι πλειάδες είναι πανομοιότυπες με μια σημαντική εξαίρεση: οι λίστες είναι μεταβαλλόμενες (mutable). Αυτό σημαίνει ότι μπορείτε να δημιουργήσετε ένα αντικείμενο λίστας και στη συνέχεια να το τροποποιήσετε στη μνήμη. Αυτό σας επιτρέπει να κάνετε πολύ ισχυρά

πράγματα σε λίστες που δεν μπορείτε να κάνετε σε πλειάδες. Και αυτές θα είναι μια από τις βασικές δομές δεδομένων που θα χρησιμοποιήσετε σε όλα τα έργα στις ψηφιακές ανθρωπιστικές επιστήμες. Ο λόγος; Συχνά πρέπει να προσαρμόσουμε τα δεδομένα ενώ τα χρησιμοποιούμε.

Όπως με τα δεδομένα, μπορούμε να δημιουργήσουμε ένα αντικείμενο λίστας στη μνήμη δημιουργώντας μια μεταβλητή ακολουθούμενη από ένα σύμβολο ίσου. Για να πούμε στην Python ότι ο συγκεκριμένος τύπος αντικειμένου που δημιουργούμε είναι μια λίστα, χρησιμοποιούμε μια αγκύλη ανοίγματος και κλεισίματος. Κάθε στοιχείο της λίστας θα χωριστεί με κόμμα. Οι λίστες μπορούν να αποθηκεύσουν οποιοδήποτε τύπο δεδομένων. Για να το δούμε αυτό στη δράση, ας δημιουργήσουμε την πρώτη μας λίστα.

```
first_list = [1, 1.0, "one"]
print(first_list)
```

```
[1, 1.0, 'one']
```

2.2.2.1. Ευρετηρίαση μιας Λίστας

Στην Python, θα χρειαστεί συχνά να έχουμε πρόσβαση σε ένα τμήμα δεδομένων μέσα σε μια λίστα ή κάποια άλλη δομή δεδομένων. Αυτό είναι γνωστό ως **ευρετηρίαση (indexing)**. Ο τρόπος με τον οποίο ευρετηριάζουμε μια λίστα είναι με μια αγκύλη ανοίγματος και κλεισίματος μέσα στην οποία τοποθετούμε τη θέση στην οποία κάθονται τα δεδομένα που θέλουμε να έχουμε πρόσβαση. Είναι σημαντικό να σημειώσουμε ότι η Python είναι μια **γλώσσα με ευρετηρίαση μηδέν (zero-index language)**. Αυτό σημαίνει ότι ξεκινάμε πάντα με τον αριθμό 0 και στη συνέχεια μετράμε προς τα πάνω, άρα το στοιχείο που βρίσκεται στην πρώτη θέση της λίστας μας είναι ευρετήριο 0.

Ας πάρουμε το στοιχείο στο ευρετήριο 0 στη first_list.

```
print(first_list[0])
```

```
1
```

Στο κελί παρακάτω, προσπαθήστε να πάρετε τη συμβολοσειρά “one” από τη first_list.

```
print(first_list)
```

```
[1, 1.0, 'one']
```

Προσέξτε ότι εκτυπώσαμε επιτυχώς τον αριθμό 1. Ωστόσο, συχνά είναι σημαντικό να ευρετηριάσουμε πολλαπλά στοιχεία σε μια λίστα. Εάν θέλουμε να το κάνουμε αυτό, χρησιμοποιούμε ξανά [ ]. Μέσα στις αγκύλες θα έχουμε μια θέση έναρξης και μια θέση λήξης. Η θέση λήξης θα είναι το σημείο μετά το οποίο θέλουμε να πάρουμε. Θα χωριστούν από :. Στον κώδικα, θα φαινόταν περίπου έτσι:

index_item[start:end]

Ας πούμε ότι θέλαμε να πάρουμε τα πρώτα 3 στοιχεία από τη λίστα, θα θέλαμε να κάνουμε κάτι τέτοιο.

```
print(first_list[0:2])
```

```
[1, 1.0]
```

Μπορούμε επίσης να εργαστούμε προς τα πίσω με την ευρετηρίαση. Μπορούμε, για παράδειγμα, να χρησιμοποιήσουμε -1 για να πάρουμε το τελευταίο στοιχείο της λίστας.

```
print(first_list[-1])
```

```
one
```

Μπορούμε επίσης να χρησιμοποιήσουμε εύρος ευρετηρίασης για να πάρουμε τα τρία τελευταία στοιχεία. Στην Python, εάν ευρετηριάσετε μια λίστα χωρίς τελικό σημείο, θα πάρει όλα όσα υπάρχουν μέχρι το τέλος αυτής της λίστας. Μπορούμε να δούμε αυτό στα δύο παραδείγματα παρακάτω.

```
print(first_list[-2:])
```

```
[1.0, 'one']
```

Μπορούμε επίσης να κάνουμε το ίδιο προς τα πίσω πιάνοντας όλα τα ευρετήρια μέχρι το πρώτο ευρετήριο. Με άλλα λόγια, το στοιχείο στο ευρετήριο 0.

```
print(first_list[:1])
```

```
[1]
```

Στην εφαρμογή Trinket παρακάτω, προσπαθήστε να δημιουργήσετε μια λίστα και στη συνέχεια να την ευρετηριάσετε σε διαφορετικά σημεία.

```
from IPython.display import IFrame
IFrame('https://trinket.io/embed/python3/3fe4c8f3f4', 700, 500)
```

2.2.3. Πλειάδες

Πλειάδες (Tuples) είναι λίστες δεδομένων που δεν μπορούν να αλλάξουν. Όταν κοιτάζουμε τις λίστες παραπάνω, θα δούμε ότι οι λίστες είναι ακριβώς το ίδιο με τις πλειάδες, εκτός ότι μπορούν να αλλάξουν. Μπορούμε να διακρίνουμε τις πλειάδες από τις λίστες από τον τρόπο με τον οποίο σχηματίζονται. Ενώ οι λίστες χρησιμοποιούν τετράγωνες αγκύλες, οι πλειάδες χρησιμοποιούν παρενθέσεις. Δημιουργούμε μια πλειάδα, όπως το παράδειγμα παρακάτω. Το αντικείμενό μας είναι first_tuple και η πλειάδα αποτελείται από τρία στοιχεία: ένας ακέραιος 1, ένας δεκαδικός 1.0 και μια συμβολοσειρά “one”. Οι λίστες και οι πλειάδες μπορούν να περιέχουν και τους τρεις αυτούς τύπους δεδομένων. Ο τρόπος με τον οποίο χωρίζουμε τα στοιχεία σε μια πλειάδα είναι με κόμμα.

```
first_tuple = (1, 1.0, "one")
```

```
print(first_tuple)
```

```
(1, 1.0, 'one')
```

Στην εφαρμογή Trinket παραπάνω, προσπαθήστε να δημιουργήσετε μια πλειάδα και να την ευρετηριάσετε.

2.2.4. Μεταβαλλόμενη vs Αμετάβλητη

Όπως σημειώθηκε παραπάνω, οι πλειάδες είναι αμετάβλητες (immutable), που σημαίνει ότι δεν μπορούν να αλλάξουν. Ας δούμε ακριβώς τι σημαίνει αυτό στην πράξη. Ας πούμε ότι θέλαμε να προσθέσουμε σε μια λίστα. Μπορούμε να το κάνουμε αυτό με τη μέθοδο .append(). Αυτή θα πάρει ένα όρισμα (argument), ή τμήμα πληροφοριών τοποθετημένο μέσα στις παρενθέσεις. Θα μάθετε για τα ορίσματα αργότερα όταν θα συζητήσουμε τις συναρτήσεις και τις μεθόδους με μεγαλύτερο βάθος. Προς το παρόν, καταλάβετε ότι οι πληροφορίες που μεταφέρονται μέσα στις παρενθέσεις λένε στη μέθοδο ή τη συνάρτηση τι χρειάζεται για να εκτελέσει τη συνάρτηση. Σε αυτή την περίπτωση, .append() μας επιτρέπει να προσθέσουμε (append) κάτι σε μια λίστα. Το όρισμα που περνάμε, “one”, λέει τι θέλουμε να προσθέσουμε. Σε αυτή την περίπτωση, η συμβολοσειρά “one”.

```
first_list.append("one")
```

```
print(first_list)
```

```
[1, 1.0, 'one', 'one']
```

Προσέξτε ότι δεν έχουμε σφάλμα. Αυτό συμβαίνει επειδή η λίστα μας είναι μεταβαλλόμενη (mutable), ή αλλάξιμη. Αυτό σημαίνει ότι μπορούμε να προσθέσουμε σε αυτήν, να διαγράψουμε στοιχεία από αυτήν και άλλες λειτουργίες που μας επιτρέπουν να αλλάξουμε τον τρόπο αποθήκευσης στη μνήμη. Οι πλειάδες, από την άλλη πλευρά, είναι αμετάβλητες (immutable), ή αναλλοίωτες. Ας προσπαθήσουμε να εκτελέσουμε την ίδια μέθοδο στη πλειάδα και ας δούμε τι συμβαίνει.

```
first_tuple.append("one")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [27], in <cell line: 1>()
----> 1 first_tuple.append("one")

AttributeError: 'tuple' object has no attribute 'append'
```

Προσέξτε ότι παίρνουμε ένα AttributeError (σφάλμα ιδιότητας). Αυτό σημαίνει ότι μια πλειάδα δεν έχει τη δυνατότητα να χρησιμοποιήσει τη μέθοδο append. Αυτό δεν υπάρχει για τις πλειάδες επειδή είναι αμετάβλητες (immutable) ή αναλλοίωτες. Ο μόνος τρόπος για να τροποποιήσουμε το όνομα του αντικειμένου, tuple1, είναι να το αντικαταστήσουμε εντελώς στη μνήμη.

Στην εφαρμογή Trinket παραπάνω, πειραματιστείτε με το append δημιουργώντας μια λίστα και προσθέτοντας δεδομένα σε αυτήν.

2.2.5. Σύνολα (Bonus Data Structure)

Υπάρχει μια άλλη δομή δεδομένων παρόμοια με λίστες και πλειάδες και την περιλαμβάνω εδώ ως μια bonus δομή δεδομένων. Αυτό είναι το σύνολο (set). Ένα **σύνολο (set)** είναι πανομοιότυπο με μια λίστα. Είναι μεταβαλλόμενη (mutable), που σημαίνει ότι μπορούμε να το ενημερώσουμε, αλλά σε αντίθεση με μια λίστα, δεν μπορεί να περιέχει διπλότυπα. Αυτό είναι χρήσιμο σε ειδικές περιστάσεις, όπως όταν πρέπει να αφαιρέσετε όλα τα διπλότυπα από μια λίστα. Το περιλαμβάνω εδώ απλώς για να γνωρίζετε ότι υπάρχουν άλλοι τύποι δομών δεδομένων.

```
first_set = {1, 1.1, "one", "one"}
print (first_set)
```

```
{1, 'one', 1.1}
```

2.2.6. Λεξικά

Όπως οι πλειάδες και οι λίστες, τα λεξικά (dictionaries) είναι μια δομή δεδομένων στην Python. Όπως και οι λίστες, τα λεξικά είναι μεταβαλλόμενα (mutable), που σημαίνει ότι μπορούν να αλλάξουν στη μνήμη. Σε αντίθεση με τις πλειάδες και τις λίστες, τα λεξικά δεν είναι λίστες δεδομένων. Αντ' αυτού, έχουν δύο συστατικά: κλειδιά (keys) και τιμές (values). Αυτά τα δύο

συστατικά χωρίζονται με κόλον. Όλα αυτά περιέχονται σε τετράγωνες παρενθέσεις. Στο παράδειγμα παρακάτω, έχουμε ένα λεξικό, a_dict, με κλειδί “name” και τιμή “William”.

```
names = {
        "first_name": "William",
         "last_name": "Mattingly"
        }
```

```
print(names)
```

```
{'first_name': 'William', 'last_name': 'Mattingly'}
```

Στα έργα στις ψηφιακές ανθρωπιστικές επιστήμες, τα λεξικά είναι ιδιαίτερα χρήσιμα για τη δόμηση πολύπλοκων δεδομένων που ενδέχεται να έχετε σε ένα Excel με κάθε κλειδί να είναι ένας στήλη του Excel και κάθε τιμή να είναι η αντίστοιχη τιμή της. Το όνομα του λεξικού θα μπορούσε να είναι το όνομα του ατόμου στο οποίο αντιστοιχεί η σειρά. Όπως οι λίστες και οι πλειάδες, μπορείτε να ενσωματώσετε δομές δεδομένων μέσα σε ένα λεξικό Python.

Ενώ θα μπορούσαμε ρεαλιστικά να αποθηκεύσουμε τα δεδομένα μας σε μια λίστα όπως παρακάτω (name_list), θα πρέπει να είμαστε συνεπείς και να τοποθετούμε πάντα το όνομα σε ευρετήριο 0 και το επώνυμο σε ευρετήριο 1. Αυτό εισάγει πιθανά προβλήματα αργότερα σε ένα έργο. Φανταστείτε εάν ένας προγραμματιστής έφυγε από ένα έργο. Χωρίς καλή τεκμηρίωση, δεν υπάρχει τίποτα εγγενές σε αυτή τη λίστα που ισοδυναμεί ευρετήριο 0 με το όνομα και ευρετήριο 1 με το επώνυμο. Εξαρτάται εξ ολοκλήρου από τον αναγνώστη των δεδομένων να κάνει νόημα αυτής της δομής δεδομένων.

Θυμηθείτε, στον προγραμματισμό είναι πάντα καλύτερο να είστε ρητοί και να παράγετε κώδικα που μπορούν να κατανοήσουν και άλλοι. Το λεξικό μας επιτρέπει να δημιουργήσουμε κλειδιά που υποδεικνύουν με μεγαλύτερη ειδικότητα τον τύπο δεδομένων με τον οποίο εργαζόμαστε. Ξέρουμε από το λεξικό names παραπάνω ότι “William” είναι ένα όνομα και “Mattingly” είναι ένα επώνυμο χωρίς να χρειάζεται να σκεφτούμε σε ποιο ευρετήριο κάθεται κάθε συμβολοσειρά. Μπορούμε να το κάνουμε αυτό επειδή τα κλειδιά του λεξικού είναι ρητά.

```
name_list = ["William", "Mattingly"]
print (name_list)
```

```
['William', 'Mattingly']
```

2.2.6.1. Ευρετηρίαση Λεξικών

Στην Python, θα πρέπει συχνά να ευρετηριάσουμε ένα λεξικό. Τα λεξικά, θυμηθείτε, είναι λίγο διαφορετικά από τις λίστες και τις πλειάδες. Αντί να είναι μια σειρά στοιχείων σε μια λίστα, ένα λεξικό είναι μια συλλογή κλειδιών και αντίστοιχων τιμών. Για να ευρετηριάσουμε ένα λεξικό, επομένως, πρέπει να εργαστούμε λίγο διαφορετικά. Αντί να ευρετηριάζουμε σε ένα συγκεκριμένο σημείο, ευρετηριάζουμε τα λεξικά σε ένα συγκεκριμένο κλειδί.

Για να κατανοήσουμε αυτό, είναι καλύτερο να το δούμε στην πράξη, ας προσπαθήσουμε να πάρουμε το όνομα στο λεξικό names.

```
print (names["first_name"])
```

```
William
```

Στην εφαρμογή Trinket παρακάτω, δημιουργήστε ένα λεξικό και πρακτικός την ευρετηρίαση του σε διαφορετικά κλειδιά.

```
IFrame('https://trinket.io/embed/python3/3fe4c8f3f4', 700, 500)
```

2.2.7. Κουίζ

▶ **Show code cell source**

```
from jupyterquiz import display_quiz
import md2json
import json
myquiz = md2json.convert(quiz)
myquiz = json.loads(myquiz)
display_quiz(myquiz)
```

What kind of data structure is made with ()?

- Dictionary
- Tuple
- List

What index does a Python list start at?

- 0
- 1

What kind of data structure is made with {}?

- Dictionary
- Tuple
- List

What are the two parts of a dictionary?

- Value
- Key
- Index

What kind of data structure is made with [ ]?

Tuple

Dictionary

List

3. Βρόχοι και Λογική

Θέματα που Καλύπτονται σε αυτό το Κεφάλαιο

1. Τι είναι Βρόχοι;
2. Επαναλήψεις
3. Γιατί είναι Σημαντικοί οι Βρόχοι;
4. Βρόχοι For
5. Βρόχοι While
6. Τι είναι οι Δηλώσεις Συνθήκης
7. Ποια είναι η Λογική Πίσω από Αυτές
8. Αν
9. Διαφορετικά
10. Ή Αν
11. Ο Τελεστής in
12. Ο Τελεστής not in

3.1. Εισαγωγή στους Βρόχους

3.1.1. Θέματα που Καλύπτονται σε αυτό το Κεφάλαιο

1. Τι είναι οι Βρόχοι;
2. Επαναλήψεις
3. Γιατί είναι Σημαντικοί οι Βρόχοι;
4. Βρόχοι For

5. Βρόχοι While

3.1.2. Τι είναι οι Βρόχοι;

Οι βρόχοι (loops) είναι μια θεμελιώδης έννοια σε όλες τις γλώσσες προγραμματισμού. Είναι ουσιαστικοί σχεδόν σε κάθε κομμάτι κώδικα που γράφω και η εμπειρία μου είναι μακριά από μοναδική. Οι βρόχοι είναι ουσιαστικοί επειδή σας επιτρέπουν να επαναλάβετε συστηματικά τα δεδομένα. **Επανάληψη (Iteration)** είναι η διαδικασία με την οποία κινείστε σε ένα σύνολο δεδομένων ή συλλογή δεδομένων. Όπως θα μάθουμε σε αυτό το κεφάλαιο, υπάρχουν δύο τύποι βρόχων στην Python:

1. Βρόχοι For (For Loops)
2. Βρόχοι While (While Loops)

Ενώ τα περισσότερα καθήκοντα μπορούν να επιτευχθούν με οποιοδήποτε βρόχο, ο τρόπος με τον οποίο τους χρησιμοποιείτε είναι διαφορετικός. Στο τέλος αυτού του κεφαλαίου, θα καταλάβετε όχι μόνο πώς να δημιουργήσετε ένα βρόχο στην Python, αλλά θα έχετε μια γενική ιδέα για το πότε να χρησιμοποιήσετε ποιο βρόχο. Καθώς κωδικοποιείτε περισσότερο και περισσότερο, θα μάθετε ότι οι προγραμματιστές τείνουν να προτιμούν ένα τύπο βρόχου έναντι του άλλου. Για μένα, προτιμώ το βρόχο for. Αν με ρωτούσατε γιατί, δεν θα μπορούσα να σας δώσω απάντηση. Ίσως στο τέλος αυτού του κεφαλαίου, και εσείς να έχετε ένα αγαπημένο!

3.1.3. Βρόχοι For

Ένας καλός τρόπος να σκεφτείτε έναν βρόχο for είναι να φανταστείτε πρώτα στο μυαλό σας μια λίστα ονομάτων:

Tom, Nancy, Drew, Steph

Παρατηρήστε ότι στη λίστα παραπάνω, έχουμε τέσσερα ονόματα. Κάθε όνομα χωρίζεται από κόμμα. Ο βρόχος for μας επιτρέπει να κινηθούμε σε αυτή τη λίστα όπου εμφανίζεται κάθε κόμμα. Ακόμα σήμερα, όταν γράφω ένα βρόχο for, λέω το εξής στο μυαλό μου:

“Για κάθε στοιχείο σε αυτή τη λίστα, κάνε κάτι.”

Ας φανταστούμε ότι θέλαμε να χρησιμοποιήσουμε την Python για να εκτυπώσουμε καθένα από τα ονόματα. Θα έλεγα στο μυαλό μου:

“Για κάθε στοιχείο σε αυτή τη λίστα, εκτύπωσε το όνομα”.

Δεν έχω γράψει ούτε ένα κομμάτι κώδικα, αλλά έχω εκφράσει ένα είδος λογικής, ή σειρά ενεργειών στην κανονική αγγλική. Αυτό είναι γνωστό ως **ψευδοκώδικας (pseudo-code)**. Ο ψευδοκώδικας είναι ένας φανταστικός τρόπος να σκεφτείτε τον κώδικα επειδή είναι ανεξάρτητος από τη γλώσσα. Εννοώ ότι ο ίδιος ψευδοκώδικας μπορεί να εφαρμοστεί τόσο στην Python όσο και στη C. Είναι χρήσιμο να σκεφτούμε τον ψευδοκώδικα επειδή εστιάζει στη λογική των δηλώσεων, όχι στον κώδικα. Καθώς γράφετε πιο σύνθετα προγράμματα, θα διαπιστώσετε ότι δεν είναι πάντα η γλώσσα προγραμματισμού που είναι δύσκολη και παράγει σφάλματα, αλλά η λογική πίσω από τον τρόπο με τον οποίο έχετε υλοποιήσει αυτή τη γλώσσα. Ο ψευδοκώδικας σας επιτρέπει να καταγράψετε πρώτα τη λογική του τι θέλετε να κάνετε, τότε μπορείτε να εστιάσετε στην εκτέλεση αυτής της λογικής.

Η παραπάνω δήλωση στα αγγλικά (“Για κάθε στοιχείο σε αυτή τη λίστα, εκτύπωσε το όνομα”.) δεν θα λειτουργήσει εάν τη γράψω σε ένα αρχείο Python, αλλά μου επιτρέπει να καταγράψω τη λογική του τι θέλω να κάνω σε χαρτί. Στη συνέχεια, θα πρέπει να γράψω αυτή τη λογική σε κανονική σύνταξη Python. Ας προσπαθήσουμε να το κάνουμε τώρα.

Πρώτα, ας δημιουργήσουμε μια λίστα ονομάτων δημιουργώντας ένα αντικείμενο με το όνομα name_list. Ίσως θέλετε επίσης να καλέσετε αυτή τη μεταβλητή nameList. Και οι δύο συμβάσεις είναι αποδεκτές.

```
name_list = ["Tom", "Nancy", "Drew", "Steph"]
```

Φοβερά! Τώρα που έχουμε τη μεταβλητή name_list, ας προσπαθήσουμε να επαναλάβουμε πάνω της, ή να περάσουμε από κάθε στοιχείο σε αυτή τη λίστα. Για να το κάνουμε αυτό στην Python, θα γράφαμε κάτι τέτοιο:

```
for name in name_list:
    print(name)
```

```
Tom
Nancy
Drew
Steph
```

Ας χωρίσουμε μια σειρά πραγμάτων που συμβαίνουν εδώ. Ξεκινάμε γράφοντας “for”. Αυτό λέει στην Python ότι πρόκειται να εισέλθουμε σε ένα βρόχο for και ότι θα πρέπει να ελέγξει τη σωστή σύνταξη. Στη συνέχεια, έχουμε τη λέξη “name”. Αυτή είναι μια μεταβλητή που θα δείχνει σε ένα αντικείμενο που δημιουργήθηκε στη μνήμη. Θα αλλάξει κάθε φορά που ο βρόχος επαναλάβει το επόμενο στοιχείο στην έκφραση, “name_list”. Αυτή η λέξη “name_list” αντιστοιχεί στο όνομα του αντικειμένου που θέλω να επαναλάβω. Τέλος, υπάρχει ένα “:”. Στην Python, αυτός είναι ο τρόπος που σημειώνουμε ένα ένθετο τμήμα κώδικα. Θα χρησιμοποιήσουμε αυτά σχεδόν σε κάθε σενάριο που γράφουμε επειδή συχνά πρέπει να ενθετεύουμε στοιχεία στον κώδικα. Μόλις η Python δει ένα “:”, θα περιμένει την επόμενη γραμμή να είναι με εσοχή.

Στην επόμενη γραμμή, έχουμε μια εσοχή ακολουθούμενη από print. Εάν θυμάστε σωστά, η συνάρτηση print μας επιτρέπει να εκτυπώσουμε κάτι στο αποτέλεσμά μας. Στη συνέχεια, έχουμε αυτό που θέλουμε να εκτυπωθεί, τη μεταβλητή “name”. Με άλλα λόγια, θέλουμε να εκτυπώσουμε κάθε όνομα που δημιουργείται προσωρινά στη μνήμη.

Παρατηρήστε στο τμήμα κώδικα παρακάτω, έχω αλλάξει το αντικείμενο “name” σε “item”. Το όνομα εδώ δεν έχει σημασία. Είναι απλώς μια λέξη που θα υποδεικνύει το όνομα του αντικειμένου που δημιουργείτε. Θεωρείται Pythonic, ή καλή μορφή, να επιλέξετε ένα όνομα για το αντικείμενο που έχει νόημα. Επειδή αυτή είναι μια name_list, έχει νόημα να χρησιμοποιήσουμε τη λέξη “name” για αυτό το όνομα αντικειμένου, ώστε άλλοι που διαβάζουν τον κώδικά σας να το καταλάβουν καλύτερα.

```
for item in name_list:
    print (item)
```

```
Tom
Nancy
Drew
Steph
```

Ενώ η δυνατότητα εκτύπωσης πραγμάτων σε ένα βρόχο for είναι χρήσιμη, ιδιαίτερα κατά τον εντοπισμό σφαλμάτων, συνήθως οι βρόχοι for χρησιμοποιούνται για τροποποίηση δεδομένων σε κάποια ικανότητα. Στο κελί παρακάτω, θέλω να είμαι σε θέση να δημιουργήσω μια νέα λίστα με βάση τη λίστα των ονομάτων στη namelist. Ωστόσο, γνωρίζω ότι όλα τα άτομα σε αυτή τη λίστα έχουν το επώνυμο “Mattingly”. Θα μπορούσα να προσθέσω χειροκίνητα το επώνυμο Mattingly σε κάθε άτομο, αλλά αυτό θα ήταν κουραστικό και, αν είχα μια λίστα χιλιάδων ονομάτων, αδύνατο.

Μέσα στο βρόχο for, επομένως, θέλω να τροποποιήσω το υπάρχον string και να το προσθέσω στη νέα λίστα, “new_names”. Μπορούμε να τροποποιήσουμε μια συμβολοσειρά με διάφορους τρόπους. Προς το παρόν, θα χρησιμοποιήσουμε την απλούστερη προσέγγιση απλής προσθήκης ενός σημείου (+) μεταξύ του ονόματος του αντικειμένου και του νέου τμήματος πληροφοριών που θέλω να προσθέσω σε αυτό, “ Mattingly”

```
new_names = []
for name in name_list:
    print (name+" Mattingly")
    new_names.append(name+" Mattingly")
```

```
Tom Mattingly
Nancy Mattingly
Drew Mattingly
Steph Mattingly
```

Τώρα που έχουμε εκτελέσει το κελί παραπάνω, ας δούμε πώς φαίνεται η νέα λίστα. Ας την εκτυπώσουμε.

```
print(new_names)
```

```
['Tom Mattingly', 'Nancy Mattingly', 'Drew Mattingly', 'Steph Mattingly']
```

Ιδού! Σαν μαγεία, έχουμε συνδυάσει δύο διαφορετικά τμήματα δεδομένων σε ένα βρόχο for.

3.1.4. Κατανόηση Λίστας

Μερικές φορές στον κώδικα κάποιου θα δείτε την ίδια ιδέα εκφρασμένη σε μια μόνο γραμμή κώδικα. Θα φαίνεται ως εξής:

```
new_names_comprehension = [name+" Mattingly" for name in name_list]
print(new_names_comprehension)
```

```
['Tom Mattingly', 'Nancy Mattingly', 'Drew Mattingly', 'Steph Mattingly']
```

Αυτό είναι γνωστό ως **κατανόηση λίστας (list comprehension)**. Η κατανόηση λίστας είναι ένας τρόπος δημιουργίας μιας λίστας σε μια μόνο γραμμή με έναν βρόχο που εμφανίζεται μέσα στις αγκύλες. Αυτή η μόνη γραμμή κάνει ακριβώς το ίδιο με τις τέσσερις γραμμές κώδικα παραπάνω, αλλά επιτρέπει σε έναν προγραμματιστή να το εκφράσει σε **σφιχτότερο κώδικα (tighter code)**. Ο σφιχτότερος κώδικας είναι μια φράση που χρησιμοποιείται συχνά για να περιγράψει κώδικα που είναι πιο συνοπτικός. Αυτό είναι συχνά αυτό που θα πρέπει να είναι ο γυαλισμένος, τελικός κώδικας και είναι ένα στυλ στο οποίο οι επιστήμονες υπολογιστών ιδιαίτερα προσπαθούν να γράψουν. Ενώ ο σφιχτότερος κώδικας θεωρείται συχνά πιο γυαλισμένος, μπορεί να είναι πιο δύσκολος για άλλους ερευνητές ή αρχάριους προγραμματιστές να καταλάβουν. Όπως και στη γραφή σε μια ομιλούμενη γλώσσα, ο σκοπός της γραφής σε κώδικα είναι επίσης να είναι αναγνώσιμος. Θα πρέπει να γνωρίζετε τους αποδέκτες σας. Εάν το κοινό σας αποτελείται από άτομα που δεν είναι επιστήμονες υπολογιστών, ίσως η γραφή πιο αναλυτικού κώδικα είναι καλύτερη.

Ας αναλύσουμε τα συστατικά αυτής της μόνης γραμμής. Πρώτα, έχουμε το αντικείμενο που θέλουμε να δημιουργήσουμε με το όνομα μεταβλητής “new_names_comprehension” θέτουμε αυτό ίσο με μια ανοιχτή αγκύλη. Μέσα σε αυτή την ανοιχτή αγκύλη δηλώνουμε τι θέλουμε να κάνουμε στη μεταβλητή που δημιουργήθηκε προσωρινά σε ολόκληρο το βρόχο. Σε αυτή την περίπτωση, αυτή η μεταβλητή είναι πάλι “name”. Ξανά, καθορίζουμε ότι θέλουμε να προσθέσουμε “ Mattingly” σε αυτό το όνομα μεταβλητής. Στη συνέχεια, καθορίζουμε το βρόχο, π.χ. “for name in name_list”. Τέλος, χρησιμοποιούμε μια κλειστή αγκύλη για να υποδείξουμε το τέλος αυτής της δήλωσης.

Προς το παρόν, εστιάστε στο να είστε πιο αναλυτικοί και να γράψετε τον κώδικά σας όπως φαίνεται στο πρώτο παράδειγμα. Αυτό θα σας βοηθήσει να κατανοήσετε τη λογική και να αποσφαλματώσετε προβλήματα καθώς εμφανίζονται πιο εύκολα, καθώς τα σφάλματά σας θα εμφανιστούν σε μια συγκεκριμένη γραμμή που προκάλεσε το σφάλμα. Όταν γίνετε πιο εξοικειωμένοι με την Python, τότε δοκιμάστε να χρησιμοποιήσετε κατανόηση λίστας.

Περιλαμβάνω κατανόηση λίστας εδώ όχι για να τη χρησιμοποιήσετε, αλλά για να την δείτε πρώτα εδώ σε ένα ελεγχόμενο περιβάλλον. Πιθανόν να δείτε αυτό σε αποθετήρια GitHub και σε StackOverflow με λίγη εξήγηση. Τώρα που έχετε δει πώς φαίνεται η κατανόηση λίστας και κατανοείτε τα στοιχεία της, θα μπορέσετε να αναλύσετε αυτόν τον κώδικα λίγο πιο εύκολα.

3.1.5. Ευρετηρίαση μιας Λίστας με και χωρίς Enumerate

Συχνά όταν δημιουργούμε έναν βρόχο, πρέπει να καταλάβουμε πού βρισκόμαστε σε ένα συγκεκριμένο ευρετήριο μιας λίστας. Όπως θα ανακαλύψετε με τον προγραμματισμό, υπάρχουν πολλοί τρόποι να το κάνετε αυτό.

Ας εξερευνήσουμε πρώτα μια απλή προσέγγιση, ώστε να κατανοήσετε τη λογική πίσω από την προσέγγιση. Στη συνέχεια, θα δούμε ένα πιο καθαρό, αλλά πιο πολύπλοκο παράδειγμα.

```
i = 0
for name in name_list:
    print(i)
    print(name)
    i=i+1
```

```
0
Tom
1
Nancy
2
Drew
3
Steph
```

Στο παραπάνω παράδειγμα, όλα είναι σχεδόν πανομοιότυπα με τον αρχικό βρόχο for με μια σαφή προσθήκη, τη μεταβλητή “i”. Το όνομα “i” εδώ αντιπροσωπεύει έναν μετρητή που αποθηκεύουμε έξω από τον βρόχο. Αρχικά θέτουμε αυτή τη μεταβλητή στο 0. Καθώς επαναλαμβάνουμε τον βρόχο, καταλήγουμε σε κάθε πάσα με τον κώδικα “i=i+1”. Αυτή η εντολή λέει στην Python ότι πάρε το αντικείμενο “i” και όποιος αριθμός είναι πρόσθεσε 1 σε αυτό. Αυτό σημαίνει ότι κάθε φορά που διέρχεσαι από τη name_list, γνωρίζεις ακριβώς πού βρίσκεσαι στο ευρετήριο.

Μπορούμε να γράψουμε αυτόν τον ακριβή κώδικα, ωστόσο, χρησιμοποιώντας τη ενσωματωμένη συνάρτηση “enumerate”. Η Enumerate δημιουργεί αυτόματα μια μεταβλητή για εμάς κατά τη διάρκεια του βρόχου και τη μετακινεί κάθε φορά που διέρχεται από ένα στοιχείο στη λίστα. Μπορούμε να χρησιμοποιήσουμε το enumerate για να γράψουμε πιο καθαρό και σφιχτότερο κώδικα.

```
for i, name in enumerate(name_list):
    print(i)
    print(name)
```

```
0
Tom
1
Nancy
2
Drew
3
Steph
```

Παρατηρήστε ότι έχουμε το ακριβώς ίδιο αποτέλεσμα όπως παραπάνω. Ας χωρίσουμε ακριβώς τι συμβαίνει σε αυτό το παράδειγμα.

Το μόνο που έχει αλλάξει είναι η ακόλουθη γραμμή:

```
for i, name in enumerate(name_list):
```

Παρατηρήστε την προσθήκη του “i, name”. Αυτό σημαίνει ότι έχουμε δύο μεταβλητές που θα δημιουργούμε κάθε φορά που κάνουμε βρόχο. Όταν χρησιμοποιείτε το enumerate, η πολύ πρώτη μεταβλητή θα πρέπει πάντα να είναι “i” ή κάτι που δείχνει στον ακέραιο που θα μετρήσει. Στη συνέχεια, έχουμε κόμμα. Αυτό χωρίζει τις δύο μεταβλητές στο βρόχο. Στη συνέχεια, έχουμε το στοιχείο στη λίστα που δημιουργούμε καθώς επαναλαμβάνουμε τα δεδομένα μας. Αυτή είναι η ίδια μεταβλητή όνομα που χρησιμοποιήσαμε πριν, “name”. Τέλος, έχουμε “enumerate(name_list):” Αυτό λέει στην Python να χρησιμοποιήσει το enumerate στη name_list. Η συνάρτηση enumerate είναι αυτό που δημιουργεί το i για εμάς στη μνήμη.

Στην περιοχή παρακάτω, δοκιμάστε να δημιουργήσετε τη δική σας λίστα και να επαναλάβετε πάνω της. Στη συνέχεια, δοκιμάστε να χρησιμοποιήσετε enumerate και να εκτυπώσετε το ευρετήριο κάθε στοιχείου στη λίστα.

▶ **Show code cell source**

3.1.6. Τελεστές

Στο Κεφάλαιο 02-01, γνωρίσαμε εν συντομία τελεστές (operators). Τους ανέφερα εκεί ώστε να ήσασταν συνειδητοί τους και επειδή συνήθως συνδέονται με αριθμούς, αλλά όπως θα δούμε, αυτοί μπορούν να χρησιμοποιηθούν σε όλους τους τύπους δεδομένων. Θα χρειαστούμε πολύ συχνά σε

βρόχους να αναγνωρίσουμε Τελεστές Σύγκρισης (ίσο με, λιγότερο από, κ.λπ.). Εδώ είναι μια λίστα αυτών:

1. Ίσο με (==)
2. Μεγαλύτερο από (>)
3. Λιγότερο από (<)
4. Λιγότερο ή ίσο με (<=)
5. Μεγαλύτερο ή ίσο με (>=)
6. Δεν είναι ίσο με (!=)

Οι τελεστές μας επιτρέπουν να επιστρέψουμε ένα Boolean (Αληθές ή Ψευδές) σχετικά με την ερώτηση που κάνουμε με αυτούς. Ας πούμε, για παράδειγμα, θέλαμε να γνωρίσουμε εάν το 1 ήταν λιγότερο από 2. Αυτό είναι Αληθές, αλλά στην Python θα μπορούσαμε να το δομήσουμε ως εξής:

```
print (1 < 2)
```

```
True
```

Θα μπορούσαμε επίσης να δομήσουμε τη ψευδή δήλωση αν το 1 ήταν μεγαλύτερο από 2.

```
print (1 > 2)
```

```
False
```

Ή ακόμα και αν το 1 ήταν ίσο με 2. Παρατηρήστε το διπλό = εδώ. Πρέπει να χρησιμοποιήσουμε δύο = επειδή ένα = στην Python δημιουργεί μια νέα μεταβλητή.

```
print (1 == 2)
```

```
False
```

Στα μαθηματικά, χρησιμοποιούμε ένα σύμβολο ίσου με διαγραφή μέσα για να δηλώσουμε δεν είναι ίσο με, αλλά στον κώδικα που δεν λειτουργεί τόσο καθαρά επειδή δεν είναι ένας χαρακτήρας στο

πληκτρολόγιο. Αντ' αυτού, χρησιμοποιούμε !=. Το πιστέψτε ή όχι, θα γίνετε πολύ καλοί γράφοντας != με την πάροδο του χρόνου. Εάν θέλαμε να δούμε αν το 1 δεν είναι ίσο με 2, θα γράφαμε αυτό:

```
print (1 != 2)
```

```
True
```

Στην επιφάνεια, ίσως να νομίζετε ότι δεν θα χρειαστεί ποτέ να χρησιμοποιήσετε τελεστές σύγκρισης. Γνωρίζετε ότι το 1 δεν είναι ίσο με 2. Στην πραγματικότητα, θα τους χρησιμοποιήσετε συνεχώς επειδή οι τελεστές σύγκρισης σας επιτρέπουν να αξιοποιήσετε τη λογική Boolean, ή τη λογική Αληθές-Ψευδές. Αυτό είναι ιδιαίτερα χρήσιμο σε βρόχους. Ένας καλός τρόπος για να δείξουμε αυτό είναι με έναν βρόχο while.

3.1.7. Βρόχοι While

Αυτοί οι τελεστές μας επιτρέπουν να δομήσουμε σύνθετες συνθήκες μέσα στο βρόχο. Έτσι, μπορούμε να πούμε ότι ενώ κάτι είναι ίσο με κάτι άλλο, η Python θα πρέπει να κάνει x. Θα δούμε αυτήν την ίδια έννοια στο επόμενο κεφάλαιο καθώς εξερευνούμε τις δήλωση συνθήκης (conditionals). Νομίζω ότι ο καλύτερος τρόπος για να μάθετε για αυτήν την έννοια είναι να πηδήσετε και να την εξερευνήσετε.

Στο βρόχο for, ο βρόχος επαναλάβαινε σε ένα σύνολο δεδομένων. Ένας βρόχος while είναι λίγο διαφορετικός. Σε έναν βρόχο while, ο βρόχος θα εκτελείται συνεχώς ενώ κάτι είναι αληθές. Όπως και στο βρόχο for, δημιουργούμε το βρόχο με ένα σύνολο εντολών που ξεκινάει με το όνομά του, while. Στη συνέχεια δηλώνουμε την συνθήκη που πρέπει να πληρωθεί και που θα έχει ως αποτέλεσμα τη διακοπή ή την αναφορά του βρόχου. Ας πούμε ότι θέλαμε να μετράμε από 0 σε 10. Θα δημιουργούσαμε ένα αντικείμενο με το όνομα x και θα το θέσουμε ίσο με τη θέση έναρξης. Στη συνέχεια θα δηλώναμε όσο x != (δεν είναι ίσο με) 10, εκτύπωσε το x. Για να διασφαλίσουμε ότι το x τσιμπάει, πρέπει να βεβαιωθούμε ότι αλλάζουμε το αντικείμενο από x σε x+1 ώστε να ανέβει κατά 1 αριθμό κάθε φορά.

```
x=0
while x != 10:
    print (x)
    x=x+1
```

```
0
1
2
3
4
5
6
7
8
9
```

Γνωρίζω ότι το έχω δηλώσει αυτό πριν, αλλά αξίζει να αναφερθεί ξανά. Στον κωδικό, συνήθως δεν υπάρχει ποτέ ένας τρόπος για να κάνετε μια εργασία. Παρατηρήστε στο τμήμα κώδικα παρακάτω τι είναι διαφορετικό; Γιατί δεν ρίχνετε μια ματιά και δείτε αν μπορείτε να καταλάβετε ακριβώς τι συμβαίνει και γιατί καταφέρνει να κάνει την ίδια εργασία;

```
x=0
while x < 10:
    print (x)
    x=x+1
```

```
0
1
2
3
4
5
6
7
8
9
```

3.1.8. Κουίζ

▶ **Show code cell source**

```
from jupyterquiz import display_quiz
import md2json
import json
myquiz = md2json.convert(quiz)
myquiz = json.loads(myquiz)
display_quiz(myquiz)
```

What are the two kinds of loops in Python?

If	While
For	Then

Which of the following is an operator?

>	<
=	!=

3.2. Δηλώσεις Συνθήκης

Όπως τα Loops, οι Δηλώσεις Συνθήκης είναι ένα απαραίτητο στοιχείο όλων των γλωσσών προγραμματισμού. **Οι δηλώσεις συνθήκης** είναι ένας τύπος λογικής στον προγραμματισμό που σας επιτρέπει να ελέγχετε εάν κάτι συμβαίνει με βάση μια συνθήκη που κάτι είναι αληθές ή ψευδές. Οι δηλώσεις συνθήκης είναι πάντα δυαδικές. Θεωρώ πάντα χρήσιμο να σκεφτώ τις δηλώσεις συνθήκης στα αγγλικά πρώτα.

Εάν κάτι είναι αληθές, τότε κάντε αυτό. Εάν εκείνο το κάτι δεν είναι αληθές, τότε μπορείτε να καθορίσετε τι θα πρέπει να συμβεί εάν αυτό συμβεί. Σε ψευδοκώδικα, θα φαινόταν κάπως έτσι:

```
if κάτι είναι αληθές:
    κάντε αυτό
if όχι:
    κάντε αυτό αντ' αυτού.
```

Παρατηρήστε πάλι την εσοχή. Καθώς προχωρούμε σε όλο αυτό το κεφάλαιο, θα μάθετε τρία απαραίτητα στοιχεία των δηλώσεων συνθήκης:

Πίνακας για Δηλώσεις Συνθήκης

κώδικας	περιγραφή
`if`	Λειτουργεί ως η αρχή μιας συνθήκης.
`else`	Λειτουργεί ως 'εάν όχι'
`elif`	Λειτουργεί ως "ή εάν".

3.2.1. Δήλωση If

Ας ξεκινήσουμε με τη δήλωση `if` δημιουργώντας πρώτα ένα αντικείμενο που ονομάζεται `x` και κάνοντας το ίσο με 0. Τώρα, μπορούμε να πούμε `εάν x είναι ίσο με 0`, τότε εκτυπώστε το. Παρατηρήστε στον κώδικα παρακάτω τα `:` και την εσοχή μετά. Όπως είδαμε με τα loops, το `:` υποδεικνύει ότι η επόμενη γραμμή πρέπει να έχει εσοχή. Κάθε φορά που έχετε μια γραμμή κώδικα που χρησιμοποιεί μια δήλωση συνθήκης, ένα `:` πρέπει πάντα να είναι στο τέλος της γραμμής για να υποδείξει στην Python πού τελειώνει αυτή η συνθήκη. Η επόμενη γραμμή επίσης, πρέπει πάντα να έχει εσοχή.

```
x=0
if x==0:
    print (x)
```

```
0
```

Έχουμε εκτυπώσει με επιτυχία τη μεταβλητή `x` επειδή η συνθήκη που δηλώσαμε είναι `Αληθής`. Ας προσπαθήσουμε να δηλώσουμε το αντίθετο αυτού. Ας θέσουμε την συνθήκη μας σε `x == 1`.

```
if x==1:
    print (x)
```

Παρατηρήστε ότι δεν συμβαίνει τίποτα. Αυτό συμβαίνει επειδή η συνθήκη μας είναι `Ψευδής`. Στην εφαρμογή Trinket παρακάτω, προσπαθήστε να δημιουργήσετε μια δήλωση συνθήκης.

3.2.2. Δήλωση Else

Εάν θέλουμε κάτι να συμβεί εάν δεν πληροί αυτή τη συνθήκη, μπορούμε να χρησιμοποιήσουμε τη δήλωση `else`. Σκεφτείτε την `else` ως "εάν όχι" στα αγγλικά. Σε ψευδοκώδικα, η `else` θα φαινόταν κάπως έτσι:

```
εάν αυτή η συνθήκη είναι Αληθής:
    κάντε αυτό
εάν όχι (else):
    κάντε κάτι άλλο
```

Ας το γράψουμε σε πραγματικό κώδικα.

```
if x==1:
    print (x)
else:
    print ("X is not 1")
```

```
X is not 1
```

Παρατηρήστε ότι η else έχει την ίδια τοποθέτηση εσοχής με τη δήλωση if και επίσης έχει ένα `:` ακολουθούμενο από κώδικα με εσοχή. Όπως σημειώθηκε παραπάνω, όλες οι γραμμές δήλωσης συνθήκης πρέπει να καταλήγουν με κόλον και η επόμενη γραμμή να έχει εσοχή.

Η δήλωση `else` είναι σημαντική επειδή επιτρέπει στον προγραμματιστή να δημιουργήσει πιο σύνθετη λογική.

3.2.3. Elif και ο τελεστής ‘in’ με Συμβολοσειρές

Δίστασα να συμπεριλάβω την `elif` σε αυτό το κεφάλαιο για φόβο ότι ενδέχεται να εισαγάγει πολλά νέα πράγματα ταυτόχρονα, αλλά νομίζω ότι είναι σημαντικό να είστε τουλάχιστον εξοικειωμένοι με την ύπαρξή της, ακόμη κι αν δεν θα τη χρησιμοποιήσετε αμέσως. Στη σύνταξη της αγγλικής γλώσσας, η `elif` λειτουργεί λίγο σαν “ή εάν”.

```
εάν η συνθήκη 1 είναι Αληθής:
    κάντε αυτό
ή εάν (elif) η συνθήκη 2 είναι Αληθής:
    κάντε αυτό αντ' αυτού
```

Εντός αυτού του παραδείγματος, λειτουργεί ελαφρώς διαφορετικά από την `else`. Σε μια δήλωση `else`, εκτελούμε μια συγκεκριμένη ενέργεια εάν η προηγούμενη συνθήκη ήταν `Ψευδής`. Με την `elif`, δηλώνουμε ότι πρέπει να ισχύει μια άλλη συνθήκη.

Στην επιφάνεια, αυτό ενδέχεται να φαίνεται ότι λειτουργεί ως δύο διαδοχικές δηλώσεις `if`, αλλά λειτουργεί ελαφρώς διαφορετικά από μια κανονική δήλωση if. Για να καταλάβετε πώς, είναι καλύτερο να δείτε τη δήλωση `elif` σε κώδικα.

Στον κώδικα παρακάτω, θα δείτε εντός της δήλωσης συνθήκης τη λέξη `in`, γνωστή ως τελεστής. Στην Python, η `in` λειτουργεί παρόμοια με τον τρόπο που λειτουργεί στα αγγλικά. Χρησιμοποιείται για να δοκιμάσει εάν κάτι είναι μέσα σε ένα τμήμα δεδομένων ή μια δομή δεδομένων. Έτσι, εάν θέλουμε να γνωρίζουμε εάν κάποια συμβολοσειρά βρίσκεται μέσα σε μια άλλη συμβολοσειρά, μπορούμε να χρησιμοποιήσουμε την `in` για να δούμε εάν αυτό συμβαίνει.

Στο παράδειγμα παρακάτω, θέλουμε να γνωρίζουμε εάν μια υποσυμβολοσειρά ή μια μικρότερη συμβολοσειρά εμφανίζεται μέσα στη μεταβλητή `text`.

```
text = "I know two people. Marge and Susan."
if "Marge" in text:
    print ("Marge Found")
elif "Susan" in text:
    print ("Susan Found")
```

```
Marge Found
```

Αυτή η έξοδος ενδέχεται να φαίνεται εκπληκτική. Τόσο η Marge όσο και η Susan είναι στη συμβολοσειρά, "text". Γιατί, λοιπόν, δεν βλέπουμε να εκτυπώνεται "Susan Found"; Ο λόγος είναι επειδή η `elif` ενεργοποιείται μόνο εάν μία από τις προηγούμενες δηλώσεις if ή άλλες δηλώσεις elif δεν έχει ενεργοποιηθεί. Εάν ήθελα να ελέγξω εάν πληροί και τις δύο συνθήκες, θα χρησιμοποιούσα, αντ' αυτού, δύο δηλώσεις `if`, όπως αυτή:

```
if "Marge" in text:
    print ("Marge Found")
if "Susan" in text:
    print ("Susan Found")
```

```
Marge Found
Susan Found
```

Η δήλωση `elif` επιτρέπει πιο ισχυρή λογική και διαδοχικές δηλώσεις συνθήκης να συμβούν. Όταν πρωτοξεκινάτε τον προγραμματισμό, ενδέχεται να μην χρησιμοποιείτε την `elif` συχνά, αλλά είναι σημαντικό, ακόμη και στις αρχές της καριέρας σας στον προγραμματισμό, να είστε εξοικειωμένοι με αυτήν.

Στην εφαρμογή Trinket παρακάτω, δημιουργήστε δηλώσεις συνθήκης.

```
IFrame("https://trinket.io/embed/python3/524597417f", 700, 356)
```

3.2.4. Δηλώσεις Συνθήκης και Λίστες με 'in' και 'not in'

Οι δηλώσεις συνθήκης είναι συχνά απαραίτητες κατά τον έλεγχο για να δούμε εάν κάτι είναι σε μια λίστα. Θυμηθείτε, οι λίστες είναι δομές δεδομένων που περιέχουν μια λίστα δεδομένων. Τα δεδομένα που περιέχονται μέσα σε μια λίστα μπορούν να είναι συμβολοσειρές, ακέραιοι, δεκαδικοί ή ακόμη και άλλες δομές δεδομένων, όπως λίστες, πλειάδες και λεξικά. Θα χρειαστεί συχνά να καταλάβετε τι περιέχει μια λίστα. Για να το κάνουμε αυτό, μπορούμε να χρησιμοποιήσουμε δηλώσεις συνθήκης με τον ίδιο τελεστή `in` που είδαμε παραπάνω.

Ας ξεκινήσουμε κάνοντας πρώτα μια λίστα που ονομάζεται `names`.

```
names = ["Terry", "Marge", "Joanne"]
```

Τώρα που έχουμε μια λίστα ονομάτων, ας χρησιμοποιήσουμε τον τελεστή `in` για να δούμε εάν το όνομα “Marge” είναι στα names.

> **Warning**
>
> Θυμηθείτε: Η Python είναι ευαίσθητη στην περίπτωση.

Σε ψευδοκώδικα, θα μπορούσαμε να γράψουμε:

```
εάν η Marge είναι στα names:
    τότε εκτυπώστε True
```

Επειδή η Python μοιάζει τόσο πολύ με τη σύνταξη της αγγλικής γλώσσας, πρέπει να κάνουμε μόνο μερικές τροποποιήσεις για να κάνει αυτό να λειτουργήσει σε πραγματικό κώδικα.

```
if "Marge" in names:
    print (True)
```

```
True
```

Το ίδιο πράγμα μπορεί επίσης να γίνει αντίστροφα. Τι γίνεται αν θέλαμε να γνωρίζουμε εάν ένα άλλο άτομο δεν ήταν στα `names`. Θα μπορούσαμε να γράψουμε αυτό ως ψευδοκώδικα που φαίνεται κάπως έτσι:

```
εάν ο Tom δεν είναι στα names:
    τότε εκτυπώστε ο Tom δεν είναι στη λίστα
```

Η Python κάνει αυτή την κατασκευή απλή επειδή ο τελεστής `in` έχει επίσης μια αρνητική εκδοχή, `not in`. Αυτός ο τελεστής λειτουργεί ακριβώς με τον ίδιο τρόπο, αλλά είναι το αντίθετο του τελεστή `in`. Ας δούμε αυτό σε δράση.

```
if "Tom" not in names:
    print ("Tom is not in the list.")
```

```
Tom is not in the list.
```

Παρατηρήστε ότι έχουμε δημιουργήσει με επιτυχία μια δήλωση συνθήκης που έλεγχε εάν μια συμβολοσειρά δεν είναι στα `names`. Και τα `in` και `not in` λειτουργούν επίσης με τον έλεγχο εάν μια υποσυμβολοσειρά εμφανίζεται σε μια συμβολοσειρά. Εάν θέλαμε να δούμε εάν μια υποσυμβολοσειρά `Jerry` εμφανίζεται στο κείμενο `Tom and Jerry`, θα μπορούσαμε να δημιουργήσουμε μια δήλωση συνθήκης που φαίνεται κάπως έτσι:

```
if "Tom" in "Tom and Jerry":
    print("Tom found.")
```

```
Tom found.
```

Στην εφαρμογή Trinket παρακάτω, πειραματιστείτε με τα `in` και `not in` και δηλώσεις συνθήκης.

3.2.5. Συμπέρασμα

Αυτό το κεφάλαιο έχει εισαγάγει τα τρία απαραίτητα στοιχεία των δηλώσεων συνθήκης: if, else και elif. Θα σας συνιστούσα να ασκηθείτε με αυτά στα δικά σας δεδομένα. Ο καλύτερος τρόπος για να μάθετε για τις δηλώσεις συνθήκης είναι να τις χρησιμοποιήσετε. Θυμηθείτε, καθώς τις γράφετε στην Python, μιλήστε δυνατά στα αγγλικά. Εξακολουθώ να μιλώ δυνατά (ή στο νου μου) ψευδοκώδικα καθώς γράφω στη σωστή σύνταξη της Python.

3.2.6. Κουίζ

```
from jupyterquiz import display_quiz
import md2json
import json
myquiz = md2json.convert(quiz)
myquiz = json.loads(myquiz)
display_quiz(myquiz)
```

Conditionals allow you to structure binary logic in your code.

- False
- True

Conditionals require you to indent.

- False
- True

What are the three words you use to create conditional statements?

- if
- then
- elif
- else

Which of the following is a good example of a conditional statement?

- if x > 0
- if x = 0:
- if x > 0:
- If x > 0:

4. Συναρτήσεις, Κλάσεις και Βιβλιοθήκες

Θέματα που Καλύπτονται σε αυτό το Κεφάλαιο

1. Τι είναι μια Συνάρτηση
2. Πώς να Δημιουργήσετε μια Συνάρτηση
3. Πώς να Ανατεθούν Ορίσματα σε Μια Συνάρτηση
4. Πώς να Ανατεθούν Ορίσματα Λέξεων-Κλειδιών σε Μια Συνάρτηση
5. Πώς να Ανατεθούν Αυθαίρετα Ορίσματα σε Μια Συνάρτηση
6. Κλάσεις
7. Μέθοδοι
8. Πώς να Δημιουργήσετε Κλάση
9. Εξωτερικές Βιβλιοθήκες
10. Πώς να Εγκαταστήσετε Βιβλιοθήκες
11. Πώς να Εισάγετε Βιβλιοθήκες

4.1. Συναρτήσεις

4.1.1. Εισαγωγή

Σε αυτήν την ενότητα, θα μάθετε τα πάντα για τις συναρτήσεις. Έχετε ήδη εκτεθεί σε αρκετές συναρτήσεις σε αυτό το εγχειρίδιο, όπως η συνάρτηση `print()`. Ο στόχος μου σε αυτήν την ενότητα είναι να σας διδάξω πώς να δημιουργείτε τις δικές σας συναρτήσεις.

Οι συναρτήσεις υπάρχουν σε όλες τις γλώσσες προγραμματισμού. Σας επιτρέπουν να γράψετε ένα σύνολο κώδικα που μπορεί να χρησιμοποιηθεί αργότερα στο πρόγραμμά σας. Είναι απαραίτητες επειδή σας επιτρέπουν να μην ξαναγράφετε το ίδιο μπλοκ κώδικα ξανά και ξανά. Ο καλός κώδικας είναι κώδικας που δεν επαναλαμβάνεται. Με άλλα λόγια, εάν το πρόγραμμά σας έχει μια εργασία που πρέπει να εκτελεί επαναλαμβανόμενα, τότε θα πρέπει να αναπτύξετε αυτόν τον κώδικα σε μια καθαρή συνάρτηση (ή κλάση που θα συναντήσουμε στην επόμενη ενότητα).

Μια συνάρτηση, όπως θα δούμε, έχει τέσσερα μέρη:

1. το όνομα της συνάρτησης

2. τα ορίσματα που περνούν στη συνάρτηση (προαιρετικό)
3. ένα docstring που εξηγεί τη συνάρτηση (προαιρετικό)
4. ο κώδικας της συνάρτησης
5. τα επιστρεφόμενα δεδομένα (προαιρετικό)

Μέχρι το τέλος αυτής της ενότητας, θα κατανοείτε καθένα από αυτά τα στοιχεία.

4.1.2. Συναρτήσεις σε Δράση

Πιστεύω ότι ο καλύτερος τρόπος για να μάθετε για τις συναρτήσεις είναι να τις δείτε σε δράση πριν αναλύσουμε τι συμβαίνει. Ας ρίξουμε μια ματιά στον παρακάτω κώδικα.

```
def adding_one(number):
    x = number+1
    return x
```

Ας αναλύσουμε τον παραπάνω κώδικα ξεκινώντας από την πρώτη γραμμή.

```
def adding_one(number):
```

Εδώ, χρησιμοποιούμε τη λέξη `def`. Στην Python, το `def` σημαίνει define (ορισμός) επειδή ορίζουμε μια συνάρτηση. Όταν η Python βλέπει το `def`, περιμένει ότι αυτό που ακολουθεί θα ακολουθήσει τη σύνταξη μιας συνάρτησης.

Το επόμενο στοιχείο στην πρώτη γραμμή είναι το `adding_one`. Αυτό είναι το όνομα της συνάρτησης. Το όνομα της συνάρτησης `print`, όπως μπορεί να περιμένετε, είναι `print`. Αυτό είναι που θα χρησιμοποιήσετε όταν θέλετε να καλέσετε τη συνάρτησή σας αργότερα στο script σας.

Στη συνέχεια, βλέπουμε `(number)`. Το όνομα μιας συνάρτησης Python πρέπει να ακολουθείται από παρενθέσεις. Μέσα στις παρενθέσεις σας, μπορείτε να τοποθετήσετε μία ή πολλές παραμέτρους. Αυτές είναι προαιρετικές. Οι παράμετροι σας επιτρέπουν να περάσετε ορίσματα σε μια συνάρτηση. Αυτά τα ορίσματα μπορούν να χρησιμοποιηθούν από τη συνάρτηση για να εκτελέσει κάποιου είδους ενέργεια είτε χειρίζοντας αυτήν την παράμετρο είτε χρησιμοποιώντας την με κάποια ικανότητα για να κάνει κάποια εργασία. Στην περίπτωση της συνάρτησής μας, η μόνη παράμετρος είναι το `number`. Εάν δημιουργείτε πολλαπλές παραμέτρους, αυτές θα πρέπει

να διαχωρίζονται με κόμμα, όπως θα δούμε παρακάτω. Παρατηρήστε ότι οι παράμετροι δεν έχουν εισαγωγικά γύρω τους. Αυτό συμβαίνει επειδή δεν είναι συμβολοσειρές, αλλά παράμετροι που θα λειτουργήσουν σαν μεταβλητές μέσα στον κώδικα της συνάρτησης.

Τέλος, σε αυτήν την πρώτη γραμμή, βλέπουμε `:`. Αυτό λέει στην Python δύο πράγματα. Πρώτον, η σύνταξη για τον ορισμό της συνάρτησης είναι πλήρης και ότι η επόμενη γραμμή θα έχει εσοχή μέσα στην οποία θα βρίσκεται ο κώδικας για τη συνάρτησή σας. Αυτό το `:` και η εσοχή είναι υποχρεωτικά.

Η επόμενη γραμμή κώδικα είναι:

```
x = number+1
```

Αυτός είναι ο κώδικας της συνάρτησης. Εδώ δημιουργούμε μια μεταβλητή μέσα στη συνάρτηση που ονομάζεται `x` η οποία θα είναι το αποτέλεσμα του `number` που περνάει στη συνάρτηση και του `1`.

Το τελευταίο τμήμα του κελιού είναι

```
return x
```

Αυτό θα επιστρέψει στον χρήστη το `x` που δημιουργήσαμε προσωρινά στη συνάρτηση.

Ας καλέσουμε τώρα αυτήν τη συνάρτηση στο παρακάτω κελί για να δούμε πώς λειτουργεί.

```
result = adding_one(1)
```

Στον παραπάνω κώδικα, έχουμε δημιουργήσει ένα νέο αντικείμενο, το `result`. Αυτό θα είναι το αποτέλεσμα της συνάρτησής μας. Όπως όλα τα αντικείμενα, μπορεί να ονομαστεί όπως θέλετε, εκτός από τα απαγορευμένα ονόματα αντικειμένων στην Python.

Στη συνέχεια καλούμε τη συνάρτηση, `adding_one` και της περνάμε ένα μόνο όρισμα, έναν ακέραιο. Σε αυτήν την περίπτωση, `1`.

Θέλω να πάρετε μια στιγμή και να προσπαθήσετε να μαντέψετε τι θα είναι το result. Προχωρήστε και προσπαθήστε να αναδημιουργήσετε αυτόν τον κώδικα στο δικό σας notebook, ή, εάν

χρησιμοποιείτε αυτό το εγχειρίδιο με το ενσωματωμένο περιβάλλον Binder, δημιουργήστε ένα νέο κελί παρακάτω και εκτυπώστε το “result”.

```
print(result)
```

▶ **Show code cell output**

Μαντέψατε σωστά; Εάν ναι, υπέροχα. Εάν όχι, δεν πειράζει. Γιατί νομίζετε ότι η απάντησή σας ήταν λάθος; Καταλαβαίνετε γιατί; Αυτές είναι μερικές από τις βασικές ερωτήσεις που θα πρέπει να θέτετε στον εαυτό σας αυτή τη στιγμή.

Εάν δώσουμε στη συνάρτηση τον αριθμό `1`, τότε η έξοδός μας θα είναι `2`. Παρομοίως, εάν το κάνουμε αυτό με το `3`, θα πάρουμε `4`. Αλλά τι θα συνέβαινε εάν προσπαθούσαμε να περάσουμε τη συμβολοσειρά `"one"`;

```
bad_result = adding_one("one")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [8], in <cell line: 1>()
----> 1 bad_result = adding_one("one")

Input In [2], in adding_one(number)
      1 def adding_one(number):
----> 2     x = number+1
      3     return x

TypeError: can only concatenate str (not "int") to str
```

Λαμβάνουμε ένα σφάλμα. Αυτό το μήνυμα σφάλματος μας επιτρέπει να κάνουμε debug το πρόβλημα. Είναι ένα `TypeError`, που σημαίνει ότι προσπαθήσαμε να χρησιμοποιήσουμε λάθος τύπο αντικειμένου. Γνωρίζουμε ότι το σφάλμα εμφανίζεται στη γραμμή 2 του κελιού μας, `x = number+1`. Γιατί νομίζετε ότι λάβαμε αυτό το σφάλμα;

Εάν είπατε επειδή το `"one"` είναι μια συμβολοσειρά και δεν μπορείτε να προσθέσετε μια συμβολοσειρά με τον ακέραιο `1`, τότε θα είχατε δίκιο. Εάν γράφαμε αυτό το πρόγραμμα για τον εαυτό μας, θα το γνωρίζαμε αυτό και δεν θα προσπαθούσαμε ποτέ να περάσουμε μια συμβολοσειρά στη συνάρτηση, αλλά τι γίνεται αν κάποιος άλλος προσπαθεί να χρησιμοποιήσει τον

κώδικά μας και δεν γνωρίζει ακριβώς τι υποτίθεται ότι κάνει; Στην Python, μπορούμε να περάσουμε κάποιες βασικές πληροφορίες για να βοηθήσουμε τους χρήστες μας.

4.1.3. Docstrings

Για να βοηθήσουμε τους χρήστες, μπορούμε να τους παρέχουμε μια μακρά συμβολοσειρά στην αρχή της συνάρτησης που εξηγεί τι κάνει η συνάρτηση. Αυτό είναι γνωστό ως `docstring`. Ας δημιουργήσουμε μια νέα συνάρτηση που ονομάζεται `adding_two` με ένα docstring. Αυτή η συνάρτηση θα κάνει το ίδιο με την προηγούμενη συνάρτησή μας, αλλά θα προσθέτει δύο στο `number` και θα επιστρέφει αυτό το αποτέλεσμα.

```
def adding_two(number):
    """
    This function expects an integer and will return that integer plus 2.
    """
    x = x+2
    return x
```

Παρατηρήστε ότι το `docstring` τυλίγεται γύρω από τρία `"` στην αρχή και στο τέλος της συμβολοσειράς. Μπορώ να έχω πρόσβαση σε αυτό το `docstring` καλώντας τη συνάρτηση και χρησιμοποιώντας το `.__doc__`.

```
print(adding_two.__doc__)
```

```
    This function expects an integer and will return that integer plus 2.
```

4.1.4. Συναρτήσεις με Πολλαπλά Ορίσματα

Στην παραπάνω ενότητα, είδαμε μια απλή συνάρτηση που είχε ένα μόνο όρισμα. Στην Python, μπορείτε να ορίσετε όσα ορίσματα θέλετε σε μια συνάρτηση. Ας προσπαθήσουμε να φτιάξουμε μια ελαφρώς διαφορετική συνάρτηση που προσθέτει δύο αριθμούς μαζί, ο καθένας παρέχεται από τον χρήστη.

```
def my_function2(number1, number2):
    x = number1+number2
    return x
```

Παρατηρήστε ότι η my_function2 είναι ακριβώς η ίδια με την παραπάνω συνάρτηση εκτός από το ότι έχουμε δύο ορίσματα, “number1” και “number2”. Επίσης, στη συνάρτηση το x είναι το αποτέλεσμα του number1 συν το number2.

Τώρα, ας προσπαθήσουμε να χρησιμοποιήσουμε αυτήν τη συνάρτηση περνώντας δύο αριθμούς σε αυτήν: 1 και 3.

```
result2 = my_function2(1,3)
print (result2)
```

```
4
```

Ωραία! Πήραμε το αποτέλεσμα που επιθυμούσαμε. Αυτά είναι γνωστά ως θεσιακά ορίσματα (positional arguments) επειδή βασιζόμαστε στη θέση του ορίσματος στο τμήμα () της συνάρτησης. Ας τροποποιήσουμε αυτήν τη συνάρτηση ξανά ελαφρώς ώστε να δείτε τι εννοώ.

```
def my_function3(number1, number2):
    print ("Number 1 is ", number1)
    print ("Number 2 is ", number2)
```

```
my_function3(1, 3)
```

```
Number 1 is  1
Number 2 is  3
```

Παρατηρήστε ότι έχω διαγράψει τη γραμμή return. Αυτό συμβαίνει επειδή αυτή η συνάρτηση δεν χρειάζεται να επιστρέψει τίποτα στον χρήστη. Αντίθετα, ο μόνος της σκοπός είναι απλώς να εκτυπώσει ποια είναι τα δύο ορίσματα. Ας αντιστρέψουμε τώρα τη σειρά των ορισμάτων μας.

```
my_function3(3, 1)
```

```
Number 1 is  3
Number 2 is  1
```

Επειδή αυτά είναι θεσιακά ορίσματα, εξαρτώμαστε από τη θέση των ορισμάτων για να τα αναθέσουμε σωστά. Μπορούμε να το παρακάμψουμε αυτό καθορίζοντας σε ποιο όρισμα θέλουμε να αναθέσουμε πράγματα, αποφεύγοντας έτσι την εξάρτηση από τη σειρά με την οποία περνάμε τα ορίσματα στη συνάρτηση. Δείτε τον παρακάτω κώδικα για να δείτε αυτό σε δράση.

```
my_function3(number2=3, number1=1)
```

```
Number 1 is  1
Number 2 is  3
```

4.1.5. Όρισμα Λέξης-Κλειδί (Keyword Argument)

Μερικές φορές όταν δημιουργούμε μια συνάρτηση, θέλουμε να είναι προαιρετικό για τον χρήστη να περάσει ένα όρισμα. Σε αυτές τις περιπτώσεις, θα δημιουργήσουμε αυτό που είναι γνωστό ως όρισμα λέξης-κλειδί (keyword argument), κάτι που έχει οριστεί σε μια προεπιλεγμένη τιμή. Στην παρακάτω συνάρτηση, θέλουμε να δώσουμε στον χρήστη την επιλογή να περάσει ένα επώνυμο στη συνάρτηση. Παρατηρήστε, ωστόσο, ότι από προεπιλογή είναι “Mattingly”.

```
def add_surname(first_name, last_name = "Mattingly"):
    print (first_name, last_name)
```

```
add_surname("William")
```

```
William Mattingly
```

Αυτό λειτούργησε καλά, αλλά ένας χρήστης έχει ακόμα τη δυνατότητα να αλλάξει το αντικείμενο last_name.

```
add_surname("William", "Smith")
```

```
William Smith
```

4.1.6. Αυθαίρετα Ορίσματα Λέξης-Κλειδί (Keyword Arbitrary Arguments)

Σε σπάνιες περιπτώσεις, δεν θα γνωρίζετε ακριβώς πόσα ορίσματα θα χρειαστεί να περάσει ένας χρήστης στη συνάρτησή σας, οπότε θέλετε να τους δώσετε την επιλογή να περάσουν όσα επιθυμούν. Σε αυτές τις περιπτώσεις, θα χρησιμοποιήσετε αυτό που είναι γνωστό ως αυθαίρετα ορίσματα (arbitrary arguments). Τα αναθέτετε αυτά με ένα * πριν από το όνομα του ορίσματος.

```
def print_names(*names):
    for name in names:
        print (name)
```

```
print_names("William", "Marge", "Sally", "Alex")
```

```
William
Marge
Sally
Alex
```

Παρατηρήστε ότι η συνάρτηση επέτρεψε στον χρήστη να περάσει όσα ορίσματα επιθυμεί στη συνάρτηση. Σε όλη τη διάρκεια του προγραμματισμού μου, έχω χρησιμοποιήσει ένα αυθαίρετο όρισμα μόνο λίγες φορές, αλλά είναι σημαντικό να το έχετε στο πίσω μέρος του μυαλού σας σε περίπτωση που βρεθείτε ποτέ σε αυτήν την κατάσταση.

4.1.7. Συμπέρασμα

Ελπίζω, τώρα να έχετε μια βασική κατανόηση για το τι είναι οι συναρτήσεις και πώς λειτουργούν στην πράξη. Συνιστώ να περάσετε μερικές ώρες παίζοντας με μερικές βασικές συναρτήσεις για να κάνετε ορισμένες εργασίες που χρειάζεστε να εκτελέσετε στα δικά σας προσωπικά δεδομένα. Εάν κολλήσετε, δοκιμάστε να κάνετε αναζήτηση στο Google για την ερώτησή σας. Θα εκπλαγείτε από το πόσες απαντήσεις είναι διαθέσιμες. Δώστε ιδιαίτερη προσοχή στις απαντήσεις του StackOverflow.

4.1.8. Απάντηση για το Result

```
print (result)
```

```
2
```

4.2. Κλάσεις

4.2.1. Εισαγωγή

Σε αυτήν την ενότητα, θα συναντήσουμε τις κλάσεις. **Οι κλάσεις** είναι κάπως σαν δομές δεδομένων, αλλά διαφέρουν με έναν σημαντικό τρόπο. Μπορούν να έχουν συναρτήσεις συνδεδεμένες σε αυτές. Όταν δημιουργείτε μια κλάση στην Python, ουσιαστικά δημιουργείτε ένα ειδικό είδος αντικειμένου δεδομένων που μπορεί να έχει συναρτήσεις. Οι συναρτήσεις που είναι ενσωματωμένες μέσα σε κλάσεις είναι γνωστές ως **μέθοδοι**. Στην πραγματικότητα έχετε ήδη συναντήσει μεθόδους. Στο κεφάλαιο 02_01 όταν μάθαμε για πρώτη φορά για τις συμβολοσειρές, μπορεί να θυμάστε ότι όταν τροποποιούσαμε τα δεδομένα, χρησιμοποιούσαμε μια “.” μετά το όνομα του αντικειμένου συμβολοσειράς ακολουθούμενη από τη συνάρτηση (μέθοδο) που θέλαμε να καλέσουμε, π.χ. str1.replace(something, something_else). Αυτό είναι που διαχωρίζει μια συνάρτηση από μια μέθοδο συντακτικά στην Python. Ενώ μια συνάρτηση καλείται από μόνη της, μια μέθοδος πρέπει να καλείται από ένα αντικείμενο κλάσης. Ενώ δεν υπάρχει εύκολος τρόπος να εξηγηθεί αυτή η διάκριση, ελπίζω ότι αυτή η εξήγηση θα βοηθηθεί από την εργασία με κλάσεις και μεθόδους σε πιο στενό επίπεδο παρακάτω.

4.2.2. Δημιουργία μιας Κλάσης

Ας προσπαθήσουμε να δημιουργήσουμε μια βασική κλάση. Η κλάση μας θα αποθηκεύει δεδομένα συγκεκριμένα σχετικά με αυτοκράτορες, οπότε ας προχωρήσουμε και ας της δώσουμε το όνομα `Emperor`. Αναμένουμε ότι κάθε αυτοκράτορας στο σύνολο δεδομένων μας θα έχει τέσσερα χαρακτηριστικά: όνομα, γέννηση, στέψη και θάνατο.

```
class Emperor:
    def __init__(self, name, birth, coronation, death):
        self.name = name
        self.birth = birth
        self.coronation = coronation
        self.death = death
```

Αυτός είναι ο τρόπος με τον οποίο θα φαίνεται αυτή η βασική κλάση. Μπορεί να είναι λίγο δύσκολο να αναλύσετε τι συμβαίνει εδώ όταν το δείτε για πρώτη φορά, οπότε ας το αναλύσουμε λίγο.

Στην πρώτη γραμμή, δηλώνουμε:

```
class Emperor:
```

Η πρώτη λέξη που βλέπουμε είναι η λέξη-κλειδί `class`. Η Python θα δει αυτό και θα περιμένει να ακολουθήσει η σύνταξη για ένα `αντικείμενο κλάσης`. Αυτό είναι μάλλον σαν το `def` της συνάρτησης που αρχίζει να ορίζει μια νέα συνάρτηση.

Το επόμενο πράγμα που βλέπουμε είναι το όνομα της κλάσης. Στην περίπτωσή μας, αυτό είναι το `Emperor`. Μετά το όνομα της κλάσης, έχουμε `()`. Αυτό ακολουθείται από `:` που ολοκληρώνει την πρώτη μας γραμμή. Αυτό επίσης σημαίνει ότι η Python θα περιμένει η επόμενη γραμμή να έχει εσοχή.

Η επόμενη γραμμή διαβάζει:

```
def __init__(self, name, birth coronation, death)
```

Αυτή είναι η δομή που θα χρησιμοποιήσετε για τις περισσότερες κλάσεις. Το def `__init__()` είναι μια ειδική μέθοδος που φορτώνει όταν δημιουργείται η instance της κλάσης. Στη συνέχεια, βλέπουμε τις παραμέτρους που θα λάβει η συνάρτηση μέσα στα `()`. Υπάρχουν πέντε παράμετροι. Η πρώτη είναι το `self` που δείχνει στην instance της ίδιας της κλάσης. Θα πρέπει πάντα να γράφετε πρώτα το `self` εδώ. Οι επόμενες τέσσερις παράμετροι είναι `name`, `birth`, `coronation` και `death`.

Η παράμετρος `self` επίσης επιτρέπει σε κάθε ένα από τα χαρακτηριστικά: `name`, `birth`, `coronation` και `death`, να συσχετίζεται με μια μοναδική instance της κλάσης. Αυτό θα γίνει πιο σαφές καθώς προχωράμε.

Οι επόμενες τέσσερις γραμμές έχουν εσοχή και διαβάζουν:

```
self.name = name
self.birth = birth
self.coronation = coronation
self.death = death
```

Αυτή η συνάρτηση θα προσαρτήσει αυτά τα χαρακτηριστικά και θα τα δεσμεύσει στην instance της κλάσης, δηλώνοντας `self.name = name` και ούτω καθεξής. Με αυτό, η κλάση μας δημιουργήθηκε. Τώρα, ας προσπαθήσουμε να φτιάξουμε ένα αντικείμενο που σχετίζεται με αυτήν την κλάση.

Ως Καρολίγγιος ερευνητής, μελέτησα τον Καρλομάγνο, έναν από τους πιο δημοφιλείς μεσαιωνικούς βασιλείς που ήταν ο δεύτερος Καρολίγγιος βασιλιάς. Γεννήθηκε το 742, στέφθηκε ως Ρωμαίος Αυτοκράτορας το 800 και πέθανε το 814. Αυτά είναι τα τέσσερα βασικά κομμάτια δεδομένων που χρειαζόμαστε για την κλάση μας.

Μπορούμε να δημιουργήσουμε το ειδικό μας αντικείμενο κλάσης Emperor για τον Καρλομάγνο γράφοντας την ακόλουθη γραμμή κώδικα.

```
charlemagne = Emperor("Charlemagne", 742, 800, 814)
```

Παρατηρήστε ότι δημιουργούμε το αντικείμενο κλάσης χρησιμοποιώντας το όνομα της κλάσης, Emperor() και περνώντας τέσσερα ορίσματα που σχετίζονται με τα τέσσερα χαρακτηριστικά: όνομα, γέννηση, στέψη, θάνατος.

Ας προσπαθήσουμε τώρα να εκτυπώσουμε αυτό το αντικείμενο κλάσης.

```
print (charlemagne)
```

```
<__main__.Emperor object at 0x7fdd6c16fc70>
```

Αυτό είναι καινούριο και πιθανώς όχι αυτό που περιμένατε. Αυτό υποδεικνύει ότι το αντικείμενο είναι ένα ειδικό αντικείμενο κλάσης. Μπορούμε να πάρουμε τα δεδομένα από αυτό το αντικείμενο κλάσης χρησιμοποιώντας την εντολή vars().

```
print(vars(charlemagne))
```

```
{'name': 'Charlemagne', 'birth': 742, 'coronation': 800, 'death': 814}
```

Μπορούμε να έχουμε πρόσβαση σε συγκεκριμένα κομμάτια δεδομένων στην κλάση μας καλώντας το αντικείμενο της κλάσης μας και στη συνέχεια χρησιμοποιώντας `.name_of_attribute`. Εάν θέλαμε να έχουμε πρόσβαση στο έτος γέννησης του Καρλομάγνου, για παράδειγμα, θα μπορούσαμε να χρησιμοποιήσουμε την ακόλουθη εντολή.

```
print(charlemagne.birth)
```

```
742
```

Σε αυτό το σημείο, μπορεί να σκέφτεστε στον εαυτό σας: "...ποιο είναι το θέμα; Μόλις έφτιαξα ένα λεξικό."

Και, σε αυτό το σημείο, θα είχατε δίκιο. Η κλάση μας, όπως έχει δομηθεί, δεν είναι πραγματικά τίποτα περισσότερο από ένα λεξικό δεδομένων αποθηκευμένο με ειδικό τρόπο. Αυτό πραγματικά δεν είναι μια καλή περίπτωση χρήσης για μια κλάση. Θυμηθείτε, αυτό που κάνει τις κλάσεις μοναδικές είναι η ικανότητα να προσαρτούν συναρτήσεις σε αυτές. Ας μάθουμε πώς να το κάνουμε αυτό τώρα!

4.2.3. Προσθήκη Συναρτήσεων σε μια Κλάση

Για να δημιουργήσουμε μια συνάρτηση μέσα σε μια κλάση, δημιουργούμε μια συνάρτηση μέσα σε αυτήν. Ας φτιάξουμε μια απλή συνάρτηση που θα εκτυπώνει μια συμβολοσειρά που δηλώνει ποιος είναι ο αυτοκράτορας και πότε γεννήθηκε αυτός ο αυτοκράτορας. Για να το κάνουμε αυτό, θα τροποποιήσουμε την κλάση ως εξής:

```
class Emperor():
    def __init__(self, name, birth, coron, death):
        self.name = name
        self.birth = birth
        self.coron = coron
        self.death = death
    def birth_date(self):
        print (f"{self.name} was born in {self.birth}")
```

Παρατηρήστε ότι τώρα έχουμε προσθέσει μια συνάρτηση μέσα στην κλάση μας που ονομάζεται `birth_date()`. Αυτή θα λάβει το self, ή όλα τα δεδομένα που σχετίζονται με αυτήν τη συγκεκριμένη κλάση. Αυτή η συνάρτηση τώρα θα εκτυπώνει αυστηρά ένα f string που θα δηλώνει πότε γεννήθηκε ένας συγκεκριμένος αυτοκράτορας.

Τώρα, απλώς χρειάζεται να δημιουργήσουμε μια νέα κλάση για τον Καρλομάγνο.

```
charlemagne = Emperor("Charlemagne", 742, 800, 814)
```

Μέσα σε αυτήν την κλάση, μπορούμε να έχουμε πρόσβαση στη μέθοδο `birth_date()`, ως εξής:

```
charlemagne.birth_date()
```

```
Charlemagne was born in 742
```

Και voilà! Έχετε τώρα δημιουργήσει την πρώτη σας κλάση που έχει μια συγκεκριμένη συνάρτηση συνδεδεμένη με αυτήν. Αν και καλύψαμε τα βασικά των κλάσεων εδώ, θα πρέπει να σημειωθεί ότι δεν καλύψαμε απαραίτητα τα πάντα. Οι κλάσεις έχουν πολύ περισσότερη πολυπλοκότητα και ευελιξία από αυτή που παρουσιάζεται εδώ· παρ' όλα αυτά, οι πληροφορίες που παρέχονται παραπάνω θα πρέπει να σας δώσουν αρκετή αίσθηση για τις κλάσεις ώστε να συνεχίσετε να προχωράτε.

Οι κλάσεις και οι συναρτήσεις είναι τα δύο δομικά στοιχεία οποιουδήποτε έργου. Σας επιτρέπουν να έχετε πιο σφιχτό, πιο καθαρό κώδικα που θα είναι πιο εύκολος στην ανάγνωση και θα φαίνεται γυαλισμένος και επαγγελματικός. Αποτρέπουν τη γραφή διπλότυπου κώδικα. Ενώ μπορεί να φαίνεται περιττό μερικές φορές να βλέπετε τον κώδικά σας μέσα από συναρτήσεις και κλάσεις, σας ενθαρρύνω ιδιαίτερα να αρχίσετε να τον βλέπετε με αυτόν τον τρόπο. Θα σας κάνει καλύτερο προγραμματιστή και είναι λιγότερο πιθανό να κάνετε λάθη στον κώδικά σας.

4.3. Βιβλιοθήκες στην Python

4.3.1. Εισαγωγή

Οι βιβλιοθήκες είναι κοινές σε όλες τις γλώσσες προγραμματισμού. Σας επιτρέπουν να εισαγάγετε μεγάλες ποσότητες κώδικα που περιέχουν συναρτήσεις και κλάσεις που μπορείτε να αξιοποιήσετε στον δικό σας κώδικα. Ένας καλός τρόπος να σκεφτείτε μια βιβλιοθήκη είναι ως την παλιά έκφραση: "μην ξαναεφεύρεις τον τροχό". Χρησιμοποιήστε αυτό που έχουν κάνει άλλοι! Ο ανοιχτός κώδικας είναι ανοιχτός για έναν λόγο! Οι άνθρωποι που φτιάχνουν αυτές τις βιβλιοθήκες το κάνουν έτσι ώστε να μην χρειάζεται να λύσετε συγκεκριμένα προβλήματα. Το έχουν κάνει για εσάς.

Σε αυτό το κεφάλαιο, θα μάθουμε πώς να εγκαθιστούμε και να εισάγουμε βιβλιοθήκες. Ο λόγος που το κάνουμε αυτό τώρα είναι επειδή το υπόλοιπο αυτού του εγχειριδίου θα απαιτήσει εξωτερικές βιβλιοθήκες, συγκεκριμένα τις pandas, requests και BeautifulSoup.

4.3.2. Πώς να Εγκαταστήσετε Βιβλιοθήκες Python

Η Python έρχεται προεγκατεστημένη με το pip. Το **Pip** είναι ένας διαχειριστής πακέτων. Κατεβάζει, εγκαθιστά και διαχειρίζεται διαφορετικές εκδόσεις λογισμικού στο μηχάνημά σας. Εάν έρχεστε σε αυτό το εγχειρίδιο από Windows, αυτή η έννοια μπορεί να είναι λίγο ξένη. Σκεφτείτε το pip ως κάτι που θα διαχειρίζεται όλες τις βιβλιοθήκες σας για εσάς. Εάν έχετε εγκαταστήσει την Python μέσω Anaconda, όπως σύστησα, τότε το pip θα τεθεί αυτόματα στο Path του συστήματός σας. Το **Path** στα Windows είναι ένας τρόπος με τον οποίο ο υπολογιστής σας μπορεί να καταλάβει εντολές για αρχεία exe που έχετε στο σύστημά σας. Εάν ανοίξετε το terminal και πληκτρολογήσετε "pip –version". Στα Windows, το terminal είναι το command prompt.

Επειδή χρησιμοποιώ το JupyterNotebook για αυτό το εγχειρίδιο, μπορώ να κάνω εντολές terminal με το "!" πριν από ένα μπλοκ κώδικα. Όταν πληκτρολογείτε "pip –version", θα πρέπει να δείτε κάτι σαν την ακόλουθη έξοδο.

```
!pip --version
```

```
pip 21.2.4 from /home/wjbmattingly/anaconda3/lib/python3.9/site-packages/pip (pythor
```

Εάν το pip λειτουργεί, τότε σημαίνει ότι μπορούμε να εγκαταστήσουμε τις βιβλιοθήκες που θα χρειαστούμε για το υπόλοιπο του εγχειριδίου. Θα χρειαστούμε τρεις βιβλιοθήκες:

- Pandas
- requests
- BeautifulSoup

Για να εγκαταστήσετε μια βιβλιοθήκη, χρησιμοποιείτε την εντολή pip “pip install library_name”, ως εξής:

!pip install pandas

Θα πρέπει να δείτε μια έξοδο παρόμοια με αυτήν. Εάν το κάνετε, τότε οι pandas έχουν εγκατασταθεί σωστά. Ας κάνουμε τώρα το ίδιο για τις requests και BeautifulSoup.

```
!pip install requests
```

```
Requirement already satisfied: requests in /home/wjbmattingly/anaconda3/lib/python3.
Requirement already satisfied: idna<4,>=2.5 in /home/wjbmattingly/anaconda3/lib/pyth
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/wjbmattingly/anaconda3
Requirement already satisfied: charset-normalizer~=2.0.0 in /home/wjbmattingly/anaco
Requirement already satisfied: certifi>=2017.4.17 in /home/wjbmattingly/anaconda3/li
```

Εάν έχετε ήδη εγκαταστήσει μια βιβλιοθήκη, η έξοδος θα φαίνεται κάπως σαν αυτήν παραπάνω.

Για την BeautifulSoup, χρειάζεται να πούμε pip install beautifulsoup4 (θα εξηγήσω γιατί αργότερα σε αυτό το εγχειρίδιο.)

```
!pip install beautifulsoup4
```

```
Collecting beautifulsoup4
  Downloading beautifulsoup4-4.10.0-py3-none-any.whl (97 kB)
     |████████████████████████████████| 97 kB 2.5 MB/s eta 0:00:011
?25hCollecting soupsieve>1.2
  Downloading soupsieve-2.3.1-py3-none-any.whl (37 kB)
Installing collected packages: soupsieve, beautifulsoup4
Successfully installed beautifulsoup4-4.10.0 soupsieve-2.3.1
```

Και αυτό είναι! Έχετε εγκαταστήσει 3 νέες βιβλιοθήκες Python! Πρέπει να μάθουμε ένα τελευταίο πράγμα, ωστόσο, πριν ολοκληρώσουμε αυτό το κεφάλαιο. Πρέπει να μάθουμε πώς να εισάγουμε βιβλιοθήκες μέσα σε ένα script Python.

4.3.3. Πώς να Εισάγετε μια Βιβλιοθήκη

Για να εισάγουμε μια βιβλιοθήκη, χρησιμοποιούμε την εντολή import library_name, ως εξής:

```
import requests
```

Μερικές φορές, όταν εργαζόμαστε με βιβλιοθήκες, υπάρχει ένας συγκεκριμένος Pythonic τρόπος για να εισάγουμε τη βιβλιοθήκη. Μια κλασική περίπτωση είναι οι pandas. Με τις pandas, τις εισάγουμε “as pd”. Το “as pd” σημαίνει ότι το όνομα στο script δεν θα είναι “pandas”, αλλά “pd”.

```
import pandas as pd
```

4.3.4. Συμπέρασμα

Αυτό είναι όλο που χρειάζεται να καλύψουμε σε αυτό το κεφάλαιο για τις βιβλιοθήκες. Θα πρέπει να αισθάνεστε λίγο πιο άνετα σχετικά με το τι είναι οι βιβλιοθήκες, γιατί είναι χρήσιμες, πώς να τις εγκαταστήσετε και πώς να τις εισάγετε. Καθώς συνεχίζουμε μέσα από τα τελευταία μέρη αυτού του εγχειριδίου, θα γίνετε πιο άνετοι με τις βιβλιοθήκες.

5. Εργασία με Εξωτερικά Δεδομένα

Θέματα που Καλύπτονται σε αυτό το Κεφάλαιο

1. Αρχεία Κειμένου
2. Ο Τελεστής With
3. Πώς να Ανοίξετε Αρχεία Κειμένου στην Python
4. Πώς να Διαβάσετε από Αρχεία Κειμένου στην Python
5. Πώς να Γράψετε σε Αρχεία Κειμένου στην Python
6. Μορφή Δεδομένων JSON

7. json.dump()
8. json.load()
9. Η Βιβλιοθήκη glob
10. Εργασία με το os
11. Περιήγηση Καταλόγων

5.1. Εργασία με Κειμενικά Δεδομένα

5.1.1. Εισαγωγή

Μέχρι τώρα, έχουμε εργαστεί αυστηρά με δεδομένα που έχουμε παραγάγει μέσα στον δικό μας κώδικα. Σπάνια θα αντιγράψετε και θα επικολλήσετε ποτέ δεδομένα σε ένα script Python. Αντίθετα, θα χρειαστεί να αλληλεπιδράσετε με εξωτερικά δεδομένα. Επιπλέον, θα χρειαστεί συχνά να αποθηκεύσετε δεδομένα με κάποιον τρόπο. Αυτό το κεφάλαιο είναι το πρώτο από δύο που σας καθοδηγούν σχετικά με το πώς να αλληλεπιδράτε με εξωτερικά δεδομένα. Θα εργαζόμαστε με κειμενικά δεδομένα σε αυτό το κεφάλαιο, ενώ στο επόμενο κεφάλαιο θα εργαστούμε με JSON, ή JavaScript Object Notation δεδομένα. Στο Μέρος 07, Κεφάλαιο 02, θα εργαστούμε με πινακοποιημένα δεδομένα.

5.1.2. Ο Τελεστής With

Ο τελεστής with στην Python είναι μια απαραίτητη εντολή που σας επιτρέπει να κάνετε κάτι στη μνήμη για όσο διάστημα ο τελεστής είναι ανοιχτός. Αυτό είναι πραγματικά σημαντικό όταν ανοίγετε αρχεία κειμένου. Υπάρχουν αρκετοί τρόποι να ανοίξετε αρχεία κειμένου στην Python, αλλά σας δείχνω μόνο αυτήν τη μέθοδο επειδή είναι η πιο Pythonic. Άλλες μέθοδοι απαιτούν να κλείσετε το αρχείο κειμένου στον κώδικα. Εάν δεν το κάνετε, θα παραμείνει ανοιχτό στη μνήμη. Εάν εργάζεστε με 10.000 αρχεία κειμένου (το έχω κάνει αυτό πριν), και ξεχάσετε να προσθέσετε μια εντολή κλεισίματος στον κώδικά σας ενώ επαναλαμβάνετε σε όλα τα 10.000 αρχεία, ο υπολογιστής σας θα μείνει χωρίς μνήμη και θα καταρρεύσει. Ο τελεστής with αποφεύγει εντελώς αυτό το πρόβλημα.

5.1.3. Πώς να Ανοίξετε ένα Αρχείο Κειμένου

Ας δούμε πώς φαίνεται σε δράση και στη συνέχεια θα αναλύσουμε τι συμβαίνει.

```
with open ('../data/sample.txt', "r") as f:
    data = f.read()
```

Η πρώτη γραμμή στον κώδικά μας είναι

```
with open (file, "r") as f:
```

Μπορούμε να δούμε ότι ξεκινάμε σαφώς με τον τελεστή “with”. Στη συνέχεια, καθορίζουμε τι πρόκειται να κάνουμε με τον τελεστή with. Σε αυτήν την περίπτωση, θα χρησιμοποιήσουμε την εντολή open. Το Open λέει στην Python να ανοίξει ένα αρχείο. Μέσα στις παρενθέσεις, περνάμε δύο ορίσματα. Το πρώτο όρισμα είναι μια συμβολοσειρά που αντιστοιχεί στο πού βρίσκεται το αρχείο. Εδώ, το αρχείο βρίσκεται στον υποφάκελό μας data και ονομάζεται “sample.txt”. Το τελικό στοιχείο αυτής της γραμμής είναι “as f”. Αυτό λέει στην Python να ανοίξει αυτό το αρχείο προσωρινά ως το όνομα αντικειμένου “f”. Η τελική άνω και κάτω τελεία, την οποία έχουμε δει πριν, υποδεικνύει ότι πρόκειται να κάνουμε κάτι με εσοχή.

Η επόμενη γραμμή έχει εσοχή επειδή αυτό είναι ένα ένθετο κομμάτι κώδικα που γίνεται μέσα στον τελεστή with. Η γραμμή διαβάζει:

```
data = f.read()
```

Το όνομα data είναι το όνομα αντικειμένου που αναθέτουμε στα περιεχόμενα του αρχείου. Στη συνέχεια, βλέπουμε f.read(). Αυτή η εντολή λέει στην Python να πάρει το αντικείμενο f, σε αυτήν την περίπτωση το προσωρινό αρχείο, να το ανοίξει και να διαβάσει τα περιεχόμενά του.

Τώρα, ας ρίξουμε μια ματιά σε αυτά τα περιεχόμενα! Μπορούμε να τα εκτυπώσουμε με την εντολή print.

```
print(data)
```

```
This is a sample of a text
This is another line of the same sample text
This is a third line.
```

Μπορούμε να δούμε ότι το αρχείο κειμένου περιέχει τρεις γραμμές. Συχνά είναι απαραίτητο να χωρίσετε τα δεδομένα εισόδου με βάση τις αλλαγές γραμμής. Μερικές φορές αυτό είναι απαραίτητο όταν το αρχείο κειμένου είναι μια λίστα δεδομένων με κάθε κομμάτι δεδομένων αποθηκευμένο σε μια νέα γραμμή. Άλλες φορές, πρέπει να καθαρίσετε τα δεδομένα έτσι ώστε μια παράγραφος ενός σαρωμένου βιβλίου να είναι μια συνεχής συμβολοσειρά κειμένου. Ένας από τους πιο εύκολους τρόπους για να το κάνετε αυτό είναι να χρησιμοποιήσετε την ενσωματωμένη μέθοδο, splitlines().

```
print(data.splitlines())
```

```
['This is a sample of a text', 'This is another line of the same sample text', 'This
```

Όπως μπορείτε να δείτε, το splitlines μας επιτρέπει να μετατρέψουμε τη συμβολοσειρά μας σε μια λίστα συμβολοσειρών που διαχωρίζονται από αλλαγές γραμμής.

5.1.4. Πώς να Γράψετε Δεδομένα σε ένα Αρχείο Κειμένου

Όπως είναι απαραίτητο να έχετε πρόσβαση σε εξωτερικά δεδομένα μέσω Python, θα βρείτε τον εαυτό σας συχνά να χρειάζεται να αποθηκεύσετε δεδομένα εκτός ενός script Python. Ας ρίξουμε μια ματιά στο πώς να το κάνουμε αυτό στο ακόλουθο μπλοκ κώδικα.

```
new_string = "This is a new string."
with open("../data/sample2.txt", "w") as f:
    f.write(new_string)
```

Αυτό το μπλοκ κώδικα φαίνεται πολύ παρόμοιο με αυτό παραπάνω, αλλά με μερικές εξαιρέσεις. Πρώτον, δημιουργήσαμε μια συμβολοσειρά που ονομάζεται new_string. Στη συνέχεια, πρέπει να ρίξουμε αυτήν τη νέα συμβολοσειρά σε ένα αρχείο κειμένου. Σε αυτήν την περίπτωση, θα την ρίξουμε στο data/sample2.txt. Παρατηρήστε ότι έχουμε αντικαταστήσει το “r” με “w”. Αυτό συμβαίνει επειδή λέμε στην Python ότι θέλουμε να γράψουμε στο αρχείο, όχι να το διαβάσουμε.

Στο κομμάτι κώδικα με εσοχή, χρησιμοποιούμε f.write(), παρά f.read(). Αυτό μας επιτρέπει να γράψουμε στο αντικείμενο f, παρά να διαβάσουμε από αυτό. Θα πάρει ένα όρισμα, τη συμβολοσειρά που θέλουμε να γράψουμε στο αρχείο. Στην περίπτωσή μας, είναι το αντικείμενο συμβολοσειράς, new_string.

5.2. Εργασία με Δεδομένα JSON

5.2.1. Εισαγωγή

Σε αυτό το κεφάλαιο, θα συναντήσετε δεδομένα JSON. Το JSON είναι μια μορφή δεδομένων που χρησιμοποιείται συχνά στο διαδίκτυο. Σημαίνει JavaScript Object Notation. Είναι ο κύριος τρόπος με τον οποίο τα δεδομένα αποθηκεύονται για ιστότοπους και καλούνται από JavaScript. Υπάρχουν δύο λόγοι για να μάθετε τις δομές δεδομένων JSON νωρίς στην καριέρα προγραμματισμού σας.

1. Το JSON αναγνωρίζεται οικουμενικά από όλους τους browsers.
2. Το JSON σας επιτρέπει να δομήσετε ιεραρχικά δεδομένα.

Τι είναι ιεραρχικά δεδομένα; Αυτά είναι δεδομένα που μπορεί να είναι ένθετα μέσα σε άλλα δεδομένα. Αυτός ο τύπος δεδομένων μερικές φορές είναι δύσκολο να αναπαρασταθεί καθαρά σε μορφή CSV. Ένας καλός τρόπος να σκεφτείτε αυτό είναι με την ακόλουθη δομή δεδομένων. Φανταστείτε ότι θέλετε να αποθηκεύσετε μια σειρά κειμένων στο Excel. Σε μια στήλη, θα είχατε ένα κείμενο και στη συνέχεια στην επόμενη στήλη, θα αποθηκεύατε μια λίστα ομιλητών. Τώρα, πώς αναπαριστάνετε τη λίστα ονομάτων; Θα μπορούσατε να το κάνετε έτσι:

Κείμενο	**Ονόματα**
Ο John και η Kathy γνωρίζουν ο ένας τον άλλον. Η Marge και ο Francois όχι.	John, Kathy, Marge, Francois

Αλλά αυτό μπορεί να γίνει ακατάστατο πολύ γρήγορα επειδή οι λίστες είναι δύσκολο να αποθηκευτούν ως δομή δεδομένων μέσα σε ένα κελί. Υπάρχουν τρόποι να το παρακάμψετε αυτό, αλλά όταν αρχίζετε να εργάζεστε με αυτόν τον τύπο δεδομένων CSV υπολογιστικά, μπορεί να γίνει περίπλοκο όσο βαθύτερες πηγαίνουν οι ιεραρχίες. Φανταστείτε αν για κάθε άτομο είχαμε μια ηλικία και έναν ρόλο. Πώς θα αποθηκεύαμε αυτά τα δεδομένα; Ίσως κάπως έτσι:

Κείμενο	Ονόματα	Ηλικία
Ο John και η Kathy γνωρίζουν ο ένας τον άλλον. Η Marge και ο Francois όχι.	John, Kathy, Marge, Francois	20, 25, 30, 35

Εάν θέλω να ενημερώσω τα δεδομένα, πρέπει να βεβαιωθώ ότι η λίστα ηλικιών αντιστοιχεί στη λίστα ονομάτων. Και πάλι, αυτό μπορεί να οδηγήσει σε προβλήματα στο μέλλον. Η λύση είναι να σταματήσουμε να χρησιμοποιούμε CSV ή Excel για να αποθηκεύουμε αυτόν τον τύπο δεδομένων και να χρησιμοποιήσουμε μια μορφή που είναι πιο ευέλικτη και ικανή να χειριστεί πράγματα όπως λίστες και ένθετα ιεραρχικά δεδομένα. Στην Python, η πιο εύκολη λύση είναι το JSON. Για να χρησιμοποιήσετε το JSON, δεν χρειάζεται να εγκαταστήσετε καμία ειδική βιβλιοθήκη. Έρχεται προκατασκευασμένο με μια ειδική βιβλιοθήκη που ονομάζεται JSON. Οι μόνες μέθοδοι που πρέπει να ξέρετε για να χρησιμοποιήσετε JSON: json.dump() και json.load(). Αλλά πρώτα, ας εισάγουμε το json.

```
import json
```

Ας δημιουργήσουμε τα παραπάνω δεδομένα ως δομή δεδομένων μέσα στην Python.

```
data = {"text": "John and Kathy both know each other. Marge and Francois do not.",
        "names": ["John", "Kathy", "Marge", "Francois"]}
data
```

```
{'text': 'John and Kathy both know each other. Marge and Francois do not.',
 'names': ['John', 'Kathy', 'Marge', 'Francois']}
```

5.2.2. json.dump()

Τώρα, ας προσπαθήσουμε να αποθηκεύσουμε αυτά τα δεδομένα εκτός της Python ως αρχείο JSON. Για να το κάνουμε αυτό, θα χρησιμοποιήσουμε τον τελεστή with open που μάθαμε στο προηγούμενο κεφάλαιο. Αντί να το ονομάσουμε .txt αρχείο, ωστόσο, θα το ονομάσουμε .json. Στη συνέχεια, θα εκτελέσουμε την εντολή json.dump(). Αυτό θα πάρει 2 βασικά ορίσματα: τα δεδομένα που θέλετε να ρίξετε στο αρχείο και το αντικείμενο στο οποίο θέλετε να ρίξετε τα δεδομένα, σε αυτήν την περίπτωση “f”. Το άλλο όρισμα λέξης-κλειδί εδώ είναι το indent. Μου αρέσει πάντα να το

χρησιμοποιώ αυτό επειδή κάνει το αρχείο JSON πιο εύκολο στην ανάγνωση. Βάζει εσοχή στα δεδομένα 4 κενά κάθε φορά που πηγαίνει βαθύτερα στην ιεραρχία.

```
with open ("data/sample_json.json", "w") as f:
    json.dump(data, f, indent=4)
```

5.2.3. json.load()

Τώρα που έχουμε ρίξει τα δεδομένα σε ένα αρχείο, ας προσπαθήσουμε να τα φορτώσουμε ξανά. Εδώ, θα ανοίξουμε το ίδιο αρχείο JSON, αλλά αυτή τη φορά ως αναγνώσιμο. Θα δημιουργήσουμε ένα νέο αντικείμενο, new_data και θα χρησιμοποιήσουμε json.load(). Αυτό θα πάρει ένα όρισμα, το αντικείμενο αρχείου από το οποίο θέλετε να φορτώσετε. Εφόσον το αρχείο JSON σας δεν είναι κατεστραμμένο, τα δεδομένα θα φορτωθούν επιτυχώς.

```
with open ("data/sample_json.json", "r") as f:
    new_data = json.load(f)
new_data
```

```
{'text': 'John and Kathy both know each other. Marge and Francois do not.',
 'names': ['John', 'Kathy', 'Marge', 'Francois']}
```

Αυτές είναι οι μόνες δύο εντολές που πρέπει να γνωρίζετε για να αρχίσετε να εργάζεστε με δεδομένα JSON στην Python. Σας ενθαρρύνω ιδιαίτερα να περάσετε μερικά λεπτά παίζοντας με αυτές τις εντολές και προσπαθώντας να αποθηκεύσετε δεδομένα με json.dump() και να φορτώσετε δεδομένα με json.load()

5.3. Εργασία με Πολλαπλά Αρχεία

5.3.1. Εισαγωγή

Συχνά, είναι απαραίτητο να ανοίγετε πολλαπλά αρχεία σε ένα script Python. Υπάρχουν πολλοί τρόποι για να το κάνετε αυτό, αλλά σε αντίθεση με τα περισσότερα εγχειρίδια Python, συνιστώ στους αρχάριους να χρησιμοποιούν τη βιβλιοθήκη που ονομάζεται glob. Το **Glob** έρχεται τυπικά με την Python, πράγμα που σημαίνει ότι δεν χρειάζεται να το εγκαταστήσετε. Ένας καλός τρόπος να

σκεφτείτε το glob είναι ως μια βιβλιοθήκη που σας επιτρέπει να κοιτάζετε μέσα σε έναν κατάλογο και να βρίσκετε όλα τα πιθανά αρχεία με βάση ορισμένες παραμέτρους.

5.3.2. Εργασία με το Glob

Η εργασία με το glob μπορεί να είναι λίγο μπερδεμένη στην αρχή, αλλά ας το αναλύσουμε. Πρώτα, πρέπει να εισάγουμε το glob.

```
import glob, os
os.chdir("..")
```

Στη συνέχεια, πρέπει να χρησιμοποιήσουμε την κλάση glob. Αυτή θα πάρει ένα όρισμα, τη συμβολοσειρά των αρχείων που θέλετε να βρείτε. Ας χρησιμοποιήσουμε τον υποφάκελο data ως παράδειγμα. Στο παράδειγμα παρακάτω, περνάμε μία συμβολοσειρά σε αυτήν την κλάση. Αυτή η συμβολοσειρά θα είναι ο υποφάκελος στον οποίο βρίσκονται τα δεδομένα ακολουθούμενος από μια κάθετο ακολουθούμενη από ένα *. Αυτό το * είναι γνωστό ως wild card (χαρακτήρας υποκατάστασης). Στην περίπτωσή μας, αναζητά όλα τα αρχεία μέσα σε αυτόν τον κατάλογο.

```
files = glob.glob("data/*")
```

Ας εκτυπώσουμε τώρα τα αρχεία για να δούμε τι υπάρχει μέσα στον φάκελο.

```
print (files)
```

```
['data/names_no_index.csv', 'data/network.csv', 'data/military-navy.csv', 'data/node
```

Παρατηρήστε ότι έχουμε πιάσει όλα τα αρχεία! Τις περισσότερες φορές, ωστόσο, θα θέλετε να πιάσετε μόνο αρχεία που είναι ενός συγκεκριμένου τύπου, π.χ. .txt ή .json. Για να το επιτύχετε αυτό μπορούμε να προσθέσουμε ένα .txt μετά το *. Αυτό θα πιάσει όλα τα αρχεία .txt.

```
files2 = glob.glob("data/*.txt")
print (files2)
```

5.3.3. Πιάσιμο Πολλαπλών Ένθετων Καταλόγων

Εάν κοιτάξετε στον υποκατάλογο data, θα παρατηρήσετε δύο ένθετους καταλόγους που ονομάζονται “other” και “other2”. Μπορούμε να πιάσουμε όλα τα αρχεία σε κάθε κατάλογο με τον ίδιο χαρακτήρα υποκατάστασης *.

```
files3 = glob.glob("data/*/*.txt")
print (files3)
```

5.3.4. Διάσχιση ενός Καταλόγου

Ενώ το glob είναι εύκολο στη χρήση, έχει ορισμένους περιορισμούς. Ένας από αυτούς είναι όταν χρειάζεται να διασχίσετε έναν κατάλογο. Φανταστείτε αν χρειαζόμασταν να πιάσουμε όλα τα txt αρχεία στο data, data/other και data/other2. Για να πιάσουμε όλα αυτά, πρέπει να διασχίσουμε όλους τους υποκαταλόγους του data και να συλλέξουμε όλα τα πιθανά αρχεία. Αυτό δεν είναι δυνατό να γίνει με το glob. Σε αυτές τις σπάνιες περιπτώσεις, θα πρέπει να είστε εξοικειωμένοι με τη βιβλιοθήκη os.

Η βιβλιοθήκη **os** μας επιτρέπει να κάνουμε πολλά πιο προηγμένα πράγματα στην Python που δεν θα καλύψω σε αυτό το εισαγωγικό εγχειρίδιο. Ένα από τα κύρια πράγματα για τα οποία θα χρησιμοποιήσετε το os είναι για να πλοηγηθείτε σε καταλόγους και να δημιουργήσετε καταλόγους. Για τους χρήστες Linux, πολλά από τη σύνταξη θα είναι οικεία, αλλά για τους χρήστες Windows μπορεί να φαίνεται λίγο ξένη. Προς το παρόν, ας εισάγουμε απλώς το os.

```
import os
```

Μόλις εισάγουμε το os, μπορούμε να χρησιμοποιήσουμε την εντολή os.walk. Αυτή θα πάρει μία συμβολοσειρά. Αυτή θα πρέπει να αντιστοιχεί στον κατάλογο από τον οποίο θέλετε να αρχίσετε τη διάσχιση. Φανταστείτε ότι αυτός είναι ο κατάλογός μας:

- data
  - other
  - other2

Θέλουμε να πάρουμε όλα τα αρχεία κειμένου στο data, other και other2.

```
walking = os.walk("data/")
print (walking)
```

Αυτή η παραπάνω έξοδος μας λέει ότι αυτό που βλέπουμε είναι ένας generator. Οι generators είναι λίγο πέρα από αυτό το εγχειρίδιο, αλλά σκεφτείτε τους ως ένα ειδικό είδος αντικειμένου που υπάρχει για μια μόνο στιγμή στη μνήμη. Όποτε βλέπετε generators, μπορείτε συνήθως να τους μετατρέψετε σε λίστα για να εργαστείτε μαζί τους. Ας τον μετατρέψουμε χρησιμοποιώντας τη συνάρτηση list() στην Python

```
walking = list(walking)
print (walking)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 walking = list(walking)
      2 print (walking)

NameError: name 'walking' is not defined
```

Αυτό μπορεί να είναι λίγο δύσκολο να αναλυθεί, οπότε ας επαναλάβουμε κάθε στοιχείο στο walking.

```
for item in walking:
    print (item)
```

```
('data/', ['other2', 'other'], ['sample_json.json', 'sample2.txt', 'sample.txt'])
('data/other2', [], ['sample4.txt'])
('data/other', [], ['sample3.txt'])
```

Όπως μπορούμε να δούμε, κάθε στοιχείο είναι μια πλειάδα με 3 μέρη:

- ριζικός κατάλογος
- υποφάκελοι
- αρχεία

Για τους σκοπούς μας, ενδιαφερόμαστε μόνο για τον ριζικό κατάλογο και το ίδιο το αρχείο. Μπορούμε να χρησιμοποιήσουμε αυτά τα δύο κομμάτια δεδομένων. Ας τροποποιήσουμε τώρα τον

βρόχο μας για να πιάσουμε αυτά τα δύο κομμάτια δεδομένων και να τα εκτυπώσουμε.

```
final_files = []
for item in walking:
    root = item[0]
    files = item[2]
    print ("This is the Root")
    print (root)
    print ("These are the Files")
    print (files)
    print ()
    print ()
```

```
This is the Root
data/
These are the Files
['sample_json.json', 'sample2.txt', 'sample.txt']


This is the Root
data/other2
These are the Files
['sample4.txt']


This is the Root
data/other
These are the Files
['sample3.txt']
```

Όπως μπορούμε να δούμε, τα αρχεία είναι μια λίστα. Μπορούμε στη συνέχεια να επαναλάβουμε τα αρχεία για να επανασυνδυάσουμε τη ρίζα με τα αρχεία για να καλλιεργήσουμε μια λίστα. Με το os, μπορούμε να το κάνουμε αυτό με το os.path.join(). Αυτό θα πάρει δύο ορίσματα, τον ριζικό κατάλογο και το αρχείο. Ας τροποποιήσουμε πάλι τον βρόχο μας. Θα εκτυπώσουμε τα αποτελέσματα.

```
for item in walking:
    root = item[0]
    files = item[2]
    for file in files:
        if file.endswith(".txt"):
             print(os.path.join(root, file))
```

```
data/sample2.txt
data/sample.txt
data/other2/sample4.txt
data/other/sample3.txt
```

Εξαιρετικά! Τώρα, μπορούμε να χρησιμοποιήσουμε αυτόν τον ίδιο ακριβώς βρόχο για να προσθέσουμε τα ονόματα των αρχείων σε μια κενή λίστα που ονομάζεται final_files.

```
final_files = []
for item in walking:
    root = item[0]
    files = item[2]
    for file in files:
        if file.endswith(".txt"):
             final_files.append(os.path.join(root, file))
print (final_files)
```

```
['data/sample2.txt', 'data/sample.txt', 'data/other2/sample4.txt', 'data/other/sampl
```

Παρατηρήστε ότι τώρα έχουμε όλα τα αρχεία .txt στον κύριο κατάλογο και σε όλους τους υποκαταλόγους. Αυτό θα είναι πολύ χρήσιμο όταν τα αρχεία σας βρίσκονται σε πολλαπλούς υποκαταλόγους μέσα σε πολλαπλούς υποκαταλόγους.

5.3.5. Συμπέρασμα

Θα συνιστούσα να παίξετε με τον κώδικα που σας παρέχεται σε αυτό το κεφάλαιο. Δεν μπορώ να τονίσω αρκετά πόσο συχνά θα χρειαστεί να εργαστείτε με πολλαπλά αρχεία σε ένα έργο Python. Είναι ίσως ένα από τα πράγματα που θα πρέπει να εργαστείτε σκληρά για να αποστηθίσετε. Με την πάροδο του χρόνου, θα γίνει πιο ενστικτώδες.

6. Εργασία με Δεδομένα στο Ιστό

Θέματα που Καλύπτονται σε αυτό το Κεφάλαιο

1. HTML
2. Ετικέτες
3. Ιδιότητες

6.1. Εισαγωγή στην HTML

6.1.1. Εισαγωγή

Σε αυτό το κεφάλαιο, θα μάθουμε για το web scraping (συλλογή δεδομένων από τον ιστό), μια από τις πιο ζωτικές δεξιότητες για έναν ερευνητή στις ψηφιακές ανθρωπιστικές επιστήμες. Το **Web scraping** είναι η διαδικασία κατά την οποία αυτοματοποιούμε την κλήση ενός server (που φιλοξενεί έναν ιστότοπο) και την ανάλυση αυτής της αίτησης που είναι ένα αρχείο HTML. Η **HTML** σημαίνει HyperText Markup Language (Γλώσσα Σήμανσης Υπερκειμένου). Είναι ο τρόπος με τον οποίο δομούνται οι ιστότοποι. Όταν κάνουμε scraping σε έναν ιστότοπο, γράφουμε κανόνες για την εξαγωγή κομματιών πληροφοριών από αυτόν με βάση το πώς αυτά τα δεδομένα είναι δομημένα μέσα στην HTML. Για να είμαστε ικανοί στο web scraping, επομένως, πρέπει να μπορούμε να κατανοούμε και να αναλύουμε την HTML.

Σε αυτή την ενότητα, θα αναλύσουμε την HTML και θα μάθετε τα πιο συνηθισμένα tags (ετικέτες), όπως τα div, p, strong και span. Θα μάθετε επίσης για τα attributes (χαρακτηριστικά) μέσα σε αυτά τα tags, όπως τα href, class και id. Είναι ζωτικής σημασίας να κατανοήσετε αυτό πριν προχωρήσετε στο επόμενο κεφάλαιο στο οποίο μαθαίνουμε πώς να κάνουμε web scraping με την Python.

6.1.2. Εμβάθυνση στην HTML

Γιατί λοιπόν είναι χρήσιμη η HTML; Η HTML, όπως και άλλες γλώσσες σήμανσης, όπως η **XML**, ή eXstensible Markup Language (Επεκτάσιμη Γλώσσα Σήμανσης), επιτρέπει στους χρήστες να δομήσουν δεδομένα μέσα σε δεδομένα. Αυτό επιτυγχάνεται με αυτό που είναι γνωστό ως tags (ετικέτες). Νομίζω ότι είναι καλύτερο να δούμε πώς φαίνεται αυτό στην πράξη. Ας εξετάσουμε ένα απλό αρχείο HTML.

```
<div>
    <p>This is a paragraph</p>
</div>
```

Παραπάνω, βλέπουμε μια πολύ απλή δομή αρχείου HTML. Στην πρώτη γραμμή αυτού του HTML, βλέπουμε `<div>`. Παρατηρήστε τη χρήση των `<` και `>`. Το άνοιγμα `<` υποδεικνύει την έναρξη ενός tag στην HTML. Ένα **tag** είναι ένας τρόπος να υποδηλώσουμε τη δομή μέσα σε ένα αρχείο HTML. Είναι ένας τρόπος να πούμε ότι αυτό που έρχεται μετά από αυτό το ένθετο κομμάτι κώδικα είναι αυτός ο τύπος δεδομένων. Μετά το `<`, βλέπουμε τη λέξη `div`. Αυτή η λέξη υποδηλώνει τον

τύπο του tag που χρησιμοποιούμε. Σε αυτήν την περίπτωση, δημιουργούμε ένα `div` tag. Αυτός είναι ένας από τους πιο συνηθισμένους τύπους tags στην HTML. Μετά το όνομα του tag, βλέπουμε `>`. Αυτό προσδιορίζει το τέλος της δημιουργίας του tag.

Στη γραμμή 2, βλέπουμε ένα ένθετο, ή με εσοχή κομμάτι HTML. Στην HTML, σε αντίθεση με την Python, η εσοχή είναι προαιρετική. Ωστόσο, είναι καλή πρακτική να χρησιμοποιούμε αλλαγές γραμμής και εσοχή στην HTML για να κάνουμε το έγγραφο πιο εύκολο στην ανάλυση για τους ανθρώπους. Η γραμμή δύο αρχίζει με ένα `p` tag. Το `p` tag στην HTML χρησιμοποιείται για να υποδηλώσει την αρχή μιας παραγράφου.

Μετά τη δημιουργία του `p tag`, βλέπουμε `This is a paragraph` (Αυτή είναι μια παράγραφος). Στην HTML, το κείμενο που βρίσκεται έξω από τα tags είναι κείμενο που εμφανίζεται στην ιστοσελίδα. Σε αυτήν την περίπτωση, το αρχείο HTML θα εμφάνιζε το κείμενο `This is a paragraph`. Αμέσως μετά από αυτό το κομμάτι κειμένου συναντάμε το πρώτο μας close tag (ετικέτα κλεισίματος). Ένα `close tag` στην HTML υποδεικνύει ότι αυτό το ένθετο κομμάτι δομής έχει τελειώσει. Στην περίπτωσή μας, το πρώτο close tag είναι `</p>`. Ξέρουμε ότι είναι ένα close tag λόγω του `</`, σε αντίθεση με το `<`.

Στην τελευταία μας γραμμή, βλέπουμε ένα close `div` tag.

6.1.3. Κατανόηση των Attributes

Ας ρίξουμε μια ματιά σε ένα άλλο μπλοκ HTML. Αυτή τη φορά, θα κάνουμε μια μικρή αλλαγή. Μπορείτε να την εντοπίσετε;

```
<div class="content">
    <p>This is a paragraph</p>
</div>
```

Εάν είπατε το τμήμα `class="content"` του open div tag, τότε θα είχατε δίκιο. Αυτό το κομμάτι είναι ενσωματωμένο μέσα στο tag και είναι γνωστό ως **attribute** (χαρακτηριστικό). Στην περίπτωσή μας, το ειδικό attribute που χρησιμοποιείται είναι ένα `class` attribute (ένας πολύ συνηθισμένος τύπος attribute). Αυτό το attribute έχει μια τιμή `content`.

Όταν κάνετε scraping σε ιστότοπους, μπορείτε να χρησιμοποιήσετε αυτά τα attributes για να καθορίσετε ποιο div tag θα πιάσετε. Υπάρχουν αρκετά συνηθισμένα attributes, συγκεκριμένα τα `class` και `id`.

6.1.4. Ανάλυση HTML με την BeautifulSoup

Τώρα που κατανοούμε λίγο για την HTML, ας αρχίσουμε να προσπαθούμε να την αναλύσουμε στην Python. Όταν **αναλύουμε** (parse) HTML, προσπαθούμε να αυτοματοποιήσουμε την αναγνώριση της δομής της HTML και να την ερμηνεύσουμε συστηματικά. Αυτή είναι η βάση για το web scraping. Για να αρχίσουμε, ας δημιουργήσουμε ένα απλό αρχείο HTML στη μνήμη.

```
html = """
<html>
    <body>
        <div class="content">
            <p>This is our first content</p>
        </div>
        <div class="other">
            <p>This is is another piece of content</p>
        </div>
    </body>
</html>
"""
```

Υπάρχουν πολλές βιβλιοθήκες Python διαθέσιμες για την ανάλυση δεδομένων HTML. Για πιο ισχυρά προβλήματα, το Selenium είναι το βιομηχανικό πρότυπο. Ενώ το Selenium είναι ισχυρό, έχει μια απότομη καμπύλη μάθησης και μπορεί να είναι προκλητικό για εκείνους που είναι νέοι στην Python, ειδικά για εκείνους που προγραμματίζουν στα Windows. Επιπλέον, τα περισσότερα προβλήματα web scraping μπορούν να λυθούν χωρίς το Selenium.

Για αυτούς τους λόγους, δεν θα χρησιμοποιήσουμε το Selenium σε αυτό το εγχειρίδιο, αλλά την BeautifulSoup. Η BeautifulSoup είναι μια ελαφριά βιβλιοθήκη Python για την ανάλυση HTML. Είναι γρήγορη και αποτελεσματική. Το πιο δύσκολο πράγμα σχετικά με την BeautifulSoup είναι να θυμάστε πώς να την εγκαταστήσετε και να την εισάγετε στην Python.

Για να εγκαταστήσετε την BeautifulSoup, πρέπει να εκτελέσετε την ακόλουθη εντολή στο terminal σας:

```
pip install beautifulsoup4
```

Μόλις εγκατασταθεί, μπορείτε στη συνέχεια να εισάγετε την BeautifulSoup με τον ακόλουθο τρόπο:

```
from bs4 import BeautifulSoup
```

Με την BeautifulSoup εισαγμένη, μπορούμε να χρησιμοποιήσουμε την κλάση `BeautifulSoup`. Αυτή η κλάση μας επιτρέπει να αναλύουμε HTML. Όπως θα δούμε σε όλο αυτό το κεφάλαιο, σπάνια θα έχετε HTML ως συμβολοσειρά μέσα στο Python script σας, αλλά προς το παρόν, καθώς ξεκινάμε, ας προσπαθήσουμε να αναλύσουμε την παραπάνω συμβολοσειρά `html` περνώντας την στην κλάση BeautifulSoup. Είναι Pythonic να ονομάζετε τη μεταβλητή BeautifulSoup σας `soup`. Εάν έχετε ένα πιο περίπλοκο script που αναλύει soup objects από πολλαπλούς ιστότοπους, μπορεί να θέλετε να είστε πιο πρωτότυποι στις συμβάσεις ονομασίας σας, αλλά για τους σκοπούς μας, το `soup` θα μας εξυπηρετήσει καλά.

```
soup = BeautifulSoup(html)
```

Με το soup object μας δημιουργημένο στη μνήμη, μπορούμε τώρα να αρχίσουμε να το εξετάζουμε. Εάν το εκτυπώσουμε, δεν θα παρατηρήσουμε κάτι ιδιαίτερο γι' αυτό. Φαίνεται σαν μια κανονική συμβολοσειρά.

```
print(soup)
```

```
<html>
<body>
<div class="content">
<p>This is our first content</p>
</div>
<div class="other">
<p>This is is another piece of content</p>
</div>
</body>
</html>
```

Ενώ μπορεί να μοιάζει με συμβολοσειρά, δεν είναι. Μπορούμε να το παρατηρήσουμε αυτό ρωτώντας την Python τι τύπος αντικειμένου είναι με τη συνάρτηση `type`

```
type(soup)
```

```
bs4.BeautifulSoup
```

Όπως μπορούμε να δούμε, αυτό είναι μια κλάση bs4.BeautifulSoup. Αυτό σημαίνει ότι ενώ μπορεί να μοιάζει με συμβολοσειρά, στην πραγματικότητα περιέχει περισσότερα δεδομένα στα οποία μπορεί να γίνει πρόσβαση. Για παράδειγμα, μπορούμε να χρησιμοποιήσουμε τη μέθοδο `.find` για να βρούμε συγκεκριμένα την πρώτη εμφάνιση ενός συγκεκριμένου tag. Η μέθοδος `find` έχει ένα υποχρεωτικό όρισμα, μια συμβολοσειρά που θα είναι το όνομα του tag που επιθυμείτε να εξαγάγετε. Ας πιάσουμε το πρώτο `div` tag.

```
first_div = soup.find("div")
print(first_div)
```

```
<div class="content">
<p>This is our first content</p>
</div>
```

Όπως μπορείτε να δείτε, καταφέραμε να πιάσουμε το πρώτο `div` tag με το `.find`, αλλά ξέρουμε ότι υπάρχουν πολλαπλά `div` tags στη συμβολοσειρά HTML μας. Τι θα γινόταν αν θέλαμε να τα πιάσουμε όλα; Γι' αυτό, μπορούμε να χρησιμοποιήσουμε τη μέθοδο `.find_all`. Όπως το `.find`, το `.find_all` παίρνει ένα μόνο υποχρεωτικό όρισμα. Και πάλι, είναι η συμβολοσειρά του ονόματος του tag που επιθυμείτε να εξαγάγετε.

```
all_divs = soup.find_all("div")
print(all_divs)
```

```
[<div class="content">
<p>This is our first content</p>
</div>, <div class="other">
<p>This is is another piece of content</p>
</div>]
```

Σε αντίθεση με το `.find` που μας επιστρέφει ένα μόνο στοιχείο, η μέθοδος `.find_all` επιστρέφει μια λίστα tags. Τι θα γινόταν αν δεν θέλαμε όλα τα tags; Τι θα γινόταν αν θέλαμε μόνο να πιάσουμε τα `div` tags με ένα συγκεκριμένο class attribute. Η BeautifulSoup μας επιτρέπει να το κάνουμε αυτό περνώντας ένα δεύτερο όρισμα στο `.find` ή `.find_all`. Αυτό το όρισμα θα είναι ένα λεξικό του οποίου τα κλειδιά θα είναι τα attributes και οι τιμές του θα είναι τα ονόματα των attributes των tags που επιθυμείτε να εξαγάγετε.

```
div_other = soup.find_all("div", {"class": "other"})
print(div_other)
```

```
[<div class="other">
<p>This is is another piece of content</p>
</div>]
```

Μόλις αποκτήσουμε το συγκεκριμένο tag που θέλουμε να εξαγάγουμε από την HTML, μπορούμε στη συνέχεια να έχουμε πρόσβαση στα ένθετα στοιχεία του. Κάθε αντικείμενο `bs4.BeautifulSoup` λειτουργεί με τον ίδιο τρόπο όπως το αρχικό soup. Περιέχει όλα τα children (ένθετα) tags. Μπορούμε, επομένως, να πιάσουμε το `p` tag που είναι ενσωματωμένο μέσα στο `div` tag που έχει ένα `class` attribute του `other` χρησιμοποιώντας το `.find("p")`.

```
paragraph = div_other[0].find("p")
print(paragraph)
```

```
<p>This is is another piece of content</p>
```

Εάν εργαζόμαστε με κειμενικά δεδομένα, ωστόσο, σπάνια θέλουμε να διατηρήσουμε την HTML, αλλά θέλουμε να εξαγάγουμε το κείμενο μέσα στην HTML. Γι' αυτό, μπορούμε να έχουμε πρόσβαση στο ακατέργαστο κείμενο που βρίσκεται μέσα στην HTML με το `.text`.

```
print(paragraph.text)
```

```
This is is another piece of content
```

Το να μπορούμε να το κάνουμε αυτό προγραμματιστικά σημαίνει ότι μπορούμε να αυτοματοποιήσουμε το scraping αρχείων HTML μέσω Python. Μπορούμε, για παράδειγμα, να εξαγάγουμε όλο το κείμενο από κάθε `div` tag στο αρχείο μας μέσω των ακόλουθων δύο γραμμών κώδικα.

```
for div in all_divs:
    print(div.text)
```

```
This is our first content

This is is another piece of content
```

6.1.5. Πώς να Βρείτε την HTML ενός Ιστότοπου

Τώρα που είστε εξοικειωμένοι γενικά με την HTML και πώς λειτουργεί, ας βουτήξουμε και ας ρίξουμε μια ματιά σε κάποια πραγματική HTML από έναν πραγματικό ιστότοπο. Στο επόμενο κεφάλαιο, θα κάνουμε web scraping στη Wikipedia, οπότε ας επικεντρωθούμε στη Wikipedia και εδώ. Εάν χρησιμοποιείτε έναν web browser που το υποστηρίζει (Chrome και Firefox), μπορείτε να εισέλθετε σε λειτουργία προγραμματιστή (developer mode). Κάθε λειτουργικό σύστημα και browser έχει διαφορετικό σύνολο πλήκτρων συντόμευσης για να το κάνετε αυτό, αλλά σε όλους μπορείτε να κάνετε δεξί κλικ στην ιστοσελίδα και να κάνετε κλικ στο “inspect”. Αυτό θα ανοίξει τη λειτουργία προγραμματιστή. Σε αυτό το σημείο του κεφαλαίου, συνιστώ ιδιαίτερα να μεταβείτε στο βίντεο καθώς θα είναι λίγο πιο εύκολο να το ακολουθήσετε.

Για αυτό το κεφάλαιο (και το επόμενο), θα εργαστούμε συγκεκριμένα με αυτήν τη σελίδα: https://en.wikipedia.org/wiki/List_of_French_monarchs

Προχωρήστε και ανοίξτε την σε μια άλλη οθόνη ή σε μια νέα καρτέλα. Αυτό το άρθρο της Wikipedia περιέχει κάποιο κείμενο, αλλά κυρίως φιλοξενεί μια λίστα όλων των Γάλλων μοναρχών, από τους Καρολιδίους μέχρι τα μέσα του 19ου αιώνα με τον Ναπολέοντα Γ'.

Όταν επιθεωρείτε αυτή τη σελίδα, θα δείτε όλη την ένθετη HTML μέσα σε αυτήν. Αφιερώστε λίγο χρόνο και εξετάστε αυτά τα tags. Στην επόμενη ενότητα, θα μάθουμε πώς να κάνουμε scraping σε αυτή τη σελίδα, αλλά προς το παρόν ρίξτε μια ματιά σε αυτό που μπορούμε να κάνουμε με μερικές βασικές εντολές στην Python.

```
import requests
from bs4 import BeautifulSoup

url = "https://en.wikipedia.org/wiki/List_of_French_monarchs"
s = requests.get(url)

soup = BeautifulSoup(s.content)
body = soup.find("div", {"id": "mw-content-text"})
for paragraph in body.find_all("p")[:5]:
    if paragraph.text.strip() != "":
        print (paragraph.text)
```

```
The monarchs of the Kingdom of France ruled from the establishment of the Kingdom of

In August 843 the Treaty of Verdun divided the Frankish realm into three kingdoms, c

Initially, the kingdom was ruled primarily by two dynasties, the Carolingians and th
```

Στο παραπάνω κελί, χρησιμοποιήσαμε δύο βιβλιοθήκες, τις requests και BeautifulSoup για να καλέσουμε τον διακομιστή Wikipedia που φιλοξενεί αυτήν τη συγκεκριμένη σελίδα. Στη συνέχεια αναζητήσαμε το div tag που περιέχει το κύριο σώμα της σελίδας. Σε αυτήν την περίπτωση, ήταν ένα div tag του οποίου το attribute “id” αντιστοιχούσε στο “mw-content-text”. Στη συνέχεια αναζήτησα όλα τα “p” tags, ή παραγράφους μέσα σε αυτό το σώμα και εκτύπωσα το κείμενο εάν το κείμενο δεν ήταν κενό. Μέχρι το τέλος του επόμενου κεφαλαίου, όχι μόνο θα κατανοήσετε τον παραπάνω κώδικα, αλλά θα μπορείτε να τον γράψετε και να τον κωδικοποιήσετε μόνοι σας. Προς το παρόν, επιθεωρήστε αυτή τη σελίδα και δείτε αν μπορείτε να βρείτε πού βρίσκεται το div tag του οποίου το id αντιστοιχεί στο “mw-content-text” στην HTML. Είναι εντάξει αν αυτό είναι δύσκολο! Δεν είναι κάτι που μπορείτε να κάνετε γρήγορα φυσικά. Χρειάζεται εξάσκηση. Ένα κόλπο για να σας βοηθήσει να ξεκινήσετε, ωστόσο, είναι να κάνετε δεξί κλικ στην περιοχή που θέλετε να κάνετε scraping και στη συνέχεια να κάνετε κλικ στο inspect.

Μόλις αισθανθείτε άνετα με αυτό, μη διστάσετε να προχωρήσετε στο επόμενο κεφάλαιο για να αρχίσετε να μαθαίνετε πώς να κάνετε web scraping!

6.2. Scraping Ιστοσελίδων με τις Requests και

BeautifulSoup

6.2.1. Εισαγωγή

Στην προηγούμενη ενότητα, μάθαμε για τα βασικά της HTML και τη βιβλιοθήκη BeautifulSoup. Δεν εργαζόμασταν, ωστόσο, με δεδομένα που βρίσκονται στον ιστό, αλλά η HTML μας ήταν αποθηκευμένη τοπικά. Σε αυτή την ενότητα, θα μάθουμε πώς να κάνουμε κλήσεις σε έναν απομακρυσμένο διακομιστή με τη βιβλιοθήκη requests και στη συνέχεια να αναλύουμε αυτά τα δεδομένα με την BeautifulSoup.

6.2.2. Requests

Η βιβλιοθήκη **requests** μας επιτρέπει να στείλουμε ένα σήμα μέσω Python σε έναν διακομιστή. Ένας καλός τρόπος να σκεφτείτε τις requests είναι ως ένας αόρατος browser που ανοίγει στο παρασκήνιο του υπολογιστή σας. Οι Requests κάνουν ακριβώς το πράγμα που κάνει ο browser σας. Στέλνουν ένα σήμα μέσω του διαδικτύου για να συνδεθούν σε μια συγκεκριμένη διεύθυνση διακομιστή. Ενώ όλοι οι διακομιστές έχουν μια μοναδική διεύθυνση IP, συχνά το διαδίκτυο συνδέει μια συγκεκριμένη και μοναδική διεύθυνση που μπορεί να χρησιμοποιηθεί ως τρόπος σύνδεσης σε έναν διακομιστή χωρίς να χρειάζεται να πληκτρολογήσετε μια διεύθυνση IP. Σε αντίθεση με τον browser σας, ωστόσο, οι requests δεν εμφανίζουν τα αποτελέσματα για να τα δείτε. Αντίθετα, λαμβάνουν το σήμα επιστροφής και απλώς αποθηκεύουν τα δεδομένα HTML στη μνήμη.

Για να αρχίσουμε να μαθαίνουμε πώς λειτουργούν οι requests, ας εισάγουμε πρώτα τη βιβλιοθήκη requests

```
import requests
```

Τώρα που έχουμε εισάγει τις requests, ας προχωρήσουμε και ας δημιουργήσουμε ένα string object που θα είναι ο ιστότοπος που θέλουμε να κάνουμε scrape. Πάντα ονομάζω αυτή τη συμβολοσειρά “url” στον κώδικά μου.

```
url = "https://en.wikipedia.org/wiki/List_of_French_monarchs"
```

Εξαιρετικά! Τώρα μπορούμε να χρησιμοποιήσουμε τη βιβλιοθήκη requests για να κάνουμε μια κλήση σε αυτή τη συγκεκριμένη σελίδα. Θα το κάνουμε αυτό μέσω της συνάρτησης get στη

βιβλιοθήκη requests. Η συνάρτηση get έχει ένα υποχρεωτικό όρισμα: τον ιστότοπο που θέλετε να ζητήσετε. Στην περίπτωσή μας, αυτή θα είναι η συμβολοσειρά “url” μας.

```
s = requests.get(url)
```

Τώρα που έχουμε δημιουργήσει ένα request object, ας ρίξουμε μια ματιά στο πώς φαίνεται αυτό.

```
print (s)
```

```
<Response [200]>
```

Στην επιφάνεια, αυτό μοιάζει σαν να μπορεί να αποτύχαμε. Τι είναι αυτό το περίεργο “response 200” και τι σημαίνει; Αυτή η συγκεκριμένη απάντηση σημαίνει ότι η προσπάθειά μας να συνδεθούμε σε έναν διακομιστή ήταν επιτυχής. Υπάρχουν πολλοί τύποι απαντήσεων, αλλά το 200 είναι αυτό που θέλουμε να δούμε. Εάν δείτε ποτέ μια απάντηση που δεν είναι 200, μπορείτε να αναζητήσετε στο Google τη συγκεκριμένη απάντηση διακομιστή και θα μάθετε τι συμβαίνει. Μερικές φορές, μια απάντηση υποδεικνύει ότι το αίτημά σας μπλοκαρίστηκε. Αυτό μπορεί να οφείλεται στο ότι ο ιστότοπος έχει μέτρα κατά του web scraping. Σε άλλες περιπτώσεις, η σελίδα μπορεί να είναι προστατευμένη, πράγμα που σημαίνει ότι βρίσκεται πίσω από ένα login. Υπάρχουν πάρα πολλά πιθανά σφάλματα που μπορεί να προκύψουν που δεν μπορώ να τα περιγράψω όλα σε αυτό το βασικό εισαγωγικό κεφάλαιο. Ωστόσο, θα σας δώσω μια λύση σε ένα πολύ κοινό πρόβλημα που μπορεί να σας επιτρέψει να ξεπεράσετε μια κοινή απάντηση 403. Για αυτή τη λύση, δείτε την τελευταία ενότητα αυτού του κεφαλαίου.

6.2.3. BeautifulSoup

Τώρα που μάθαμε πώς να κάνουμε μια κλήση σε έναν διακομιστή και αποθηκεύσαμε την απάντηση (την HTML) στη μνήμη, χρειαζόμαστε έναν τρόπο να αναλύσουμε αυτά τα δεδομένα. Θαμμένο μέσα στο s request object είναι το περιεχόμενο HTML. Μπορούμε να έχουμε πρόσβαση σε αυτά τα δεδομένα προσπελαύνοντας τη μέθοδο content της κλάσης response object. Ας το κάνουμε αυτό και ας ελέγξουμε τους πρώτους 100 χαρακτήρες.

```
print (s.content[:100])
```

```
b'<!DOCTYPE html>\n<html class="client-nojs" lang="en" dir="ltr">\n<head>\n<meta cha
```

Παρατηρήστε ότι αυτά τα δεδομένα είναι HTML. Σε αυτό το στάδιο, ωστόσο, δεν έχουμε έναν εύκολο τρόπο να πάρουμε αυτή τη συμβολοσειρά και να την επεξεργαστούμε ως δομημένα δεδομένα. Εδώ μπαίνει στο παιχνίδι η BeautifulSoup. Η BeautifulSoup μας επιτρέπει να μετατρέψουμε το s.content σε δομημένα δεδομένα που μπορούμε στη συνέχεια να αναλύσουμε. Για να το κάνουμε αυτό, πρέπει πρώτα να εισάγουμε την BeautifulSoup. Σε αντίθεση με τις περισσότερες βιβλιοθήκες, η BeautifulSoup εγκαθίσταται ως bs4 (BeautifulSoup4). Εξαιτίας αυτού πρέπει να εισάγουμε την κλάση BeautifulSoup από το bs4. Η παρακάτω εντολή το κάνει αυτό για εμάς.

```
from bs4 import BeautifulSoup
```

Στη συνέχεια, πρέπει να δημιουργήσουμε ένα νέο soup object.

```
soup = BeautifulSoup(s.content)
```

Εάν δεν δούμε ένα σφάλμα, τότε σημαίνει ότι έχουμε δημιουργήσει επιτυχώς ένα soup object. Ας το εκτυπώσουμε για να δούμε πώς φαίνεται:

```
print (str(soup)[:200])
```

```
<!DOCTYPE html>
<html class="client-nojs" dir="ltr" lang="en">
<head>
<meta charset="utf-8"/>
<title>List of French monarchs - Wikipedia</title>
<script>document.documentElement.className="client-js";
```

Ενώ το soup object φαίνεται ακριβώς το ίδιο με το s.content, είναι εντελώς διαφορετικό. Διατηρεί τη δομή της HTML επειδή η BeautifulSoup την έχει αναλύσει για εμάς. Αυτό σημαίνει ότι μπορούμε να χρησιμοποιήσουμε ειδικές μεθόδους. Σε αυτήν την ενότητα του εγχειριδίου, θα χρησιμοποιήσουμε τις find και find_all.

- find - αυτό θα μας επιτρέψει να βρούμε την πρώτη εμφάνιση ενός tag που χρησιμοποιείται και να πιάσουμε αυτό το tag και όλα τα ένθετα στοιχεία του.

- find_all - αυτό θα επιστρέψει μια λίστα όλων των tags και των ένθετων στοιχείων τους που ευθυγραμμίζονται με αυτό το συγκεκριμένο tag.

Ας ρίξουμε μια ματιά σε ένα βασικό παράδειγμα της μεθόδου find.

```
div = soup.find("div")
```

Εδώ, πιάσαμε το πρώτο div tag στη σελίδα. Τα Div tags, ωστόσο, είναι αρκετά συνηθισμένα επειδή είναι ένα από τα βασικά δομικά στοιχεία της HTML. Ας ρίξουμε μια ματιά στο πόσα υπάρχουν σε ολόκληρη τη σελίδα. Μπορούμε να το κάνουμε αυτό με τη μέθοδο find_all και στη συνέχεια να μετρήσουμε το μέγεθος της λίστας με τη συνάρτηση len που συναντήσαμε στο κεφάλαιο 02_02.

```
all_divs = soup.find_all("div")
print (len(all_divs))
```

```
97
```

Έτσι, αν θέλουμε να πιάσουμε ένα συγκεκριμένο div μόνο με τις find και find_all, θα έπρεπε να μετρήσουμε όλα τα div tags και να βρούμε το σωστό index και να το πιάσουμε. Αυτό δεν θα λειτουργούσε σε κλίμακα επειδή αυτό θα ποικίλλει από σελίδα σε σελίδα, ακόμη και αν η δομή δεδομένων HTML ήταν παρόμοια σε όλες τις σελίδες σε έναν ιστότοπο. Χρειαζόμαστε μια καλύτερη λύση. Εδώ μπαίνει το δεύτερο προαιρετικό όρισμά μας. Η συνάρτηση find μπορεί επίσης να πάρει ένα λεξικό που μας επιτρέπει να περάσουμε κάποια συγκεκριμενοποίηση στην ανάλυσή μας του soup object.

Ας πούμε ότι θέλω να πιάσω το κύριο σώμα του άρθρου της Wikipedia. Εάν επιθεωρήσω τη σελίδα, θα παρατηρήσω ότι ένα συγκεκριμένο div tag περιέχει όλα τα δεδομένα που αντιστοιχούν στο σώμα του άρθρου της Wikipedia. Αυτό το div tag έχει ένα ειδικό μοναδικό id attribute. Το όνομα αυτού του id attribute είναι “mw-content-text”. Αυτό σημαίνει ότι μπορώ να περάσω ως δεύτερο όρισμα ένα λεξικό όπου το id είναι το κλειδί και το αντίστοιχο όνομα id είναι η τιμή. Αυτό θα πει στην BeautifulSoup να βρει το πρώτο div tag που έχει ένα id attribute που ταιριάζει με το mw-content-text. Ας ρίξουμε μια ματιά σε αυτό στον κώδικα.

```
body = soup.find("div", {"id": "mw-content-text"})
```

Τώρα που πιάσαμε αυτό το τμήμα του σώματος του άρθρου, μπορούμε να το εκτυπώσουμε με τη μέθοδο text.

```
print (body.text[:500])
```

```
This article is about French monarchs. For Frankish kings, see List of Frankish king


 Division of the Frankish Empire at the Treaty of Verdun in 843
The monarchs of the Kingdom of France ruled from the establishment of the Kingdom of
```

Αυτό είναι φανταστικό! Αλλά, τι θα γινόταν αν θέλαμε να πιάσουμε το κείμενο και να διατηρήσουμε τη δομή των παραγράφων. Μπορούμε να το κάνουμε αυτό αναζητώντας το soup object στο επίπεδο του body. Το body object που δημιουργήσαμε είναι ακόμα ένα soup object που διατηρεί όλα αυτά τα δεδομένα HTML, αλλά περιέχει μόνο τα δεδομένα για τα δεδομένα που βρίσκονται κάτω από αυτό το συγκεκριμένο div. Μπορούμε να χρησιμοποιήσουμε το find all τώρα για να πιάσουμε όλες τις παραγράφους από μέσα στο body. Μπορούμε να χρησιμοποιήσουμε το find_all στο body object για να βρούμε όλες τις παραγράφους έτσι:

```
paragraphs = body.find_all("p")
```

Μπορούμε τώρα να επαναλάβουμε τις παραγράφους. Ας το κάνουμε αυτό τώρα στις πρώτες πέντε παραγράφους και ας εκτυπώσουμε το κείμενο.

```
for paragraph in paragraphs[:5]:
    print (paragraph.text)
```

```
The monarchs of the Kingdom of France ruled from the establishment of the Kingdom of

In August 843 the Treaty of Verdun divided the Frankish realm into three kingdoms, c

Initially, the kingdom was ruled primarily by two dynasties, the Carolingians and th
```

Viola! Τώρα έχετε επίσημα κάνει webscraping στην πρώτη σας σελίδα στην Python και πιάσατε κάποια σχετικά δεδομένα. Το scraping δεδομένων από τον ιστό δεν είναι ποτέ μια εργασία αντιγραφής και επικόλλησης επειδή κάθε ιστότοπος δομεί την HTML του λίγο διαφορετικά.

Παρόλα αυτά, οι μηχανισμοί είναι οι ίδιοι. Οι βασικές μέθοδοι που μάθατε σε αυτό το κεφάλαιο θα πρέπει να σας επιτρέψουν να κάνετε scraping στην πλειοψηφία των στατικών ιστότοπων.