WAAT – Tutorato

- TUTTI I GIOVEDÌ DALLE 10:00 ALLE 12:00
- AULA 5 CM ECONOMIA
- LODO.MARCHESI@GMAIL.COM
- OGGETTO MAIL [WAAT]

Cosa vedremo?

- Phyton
- Information Retrieval
- Natural Language Tool Kit (NLTK)
- Weka
- Regex
- Ditemelo voi!
- ....

Cosa vedremo oggi

- Ripasso Python
- Chiarimento Virtual Environments e PyCharm
- Git
- 01-Esercitazione

Python- indentazione

- in Python, l’indentazione è significativa, indentare in modo incorretto può portare a comportamenti sbagliati del programma o a errori
- Esempio:

```
print("eseguito sempre all'inizio")
if condizione:
    print('eseguito in mezzo solo se la condizione è vera')
print('eseguito sempre alla fine’)
```

Python - tipi di dato

- Numeri (interi, float, complessi, ...)
- Booleani (vero o falso)
- Stringhe (rappresentano del testo)
- Liste (Una sequenza mutabile di oggetti)
- Tuple (Una sequenza immutabile di oggetti)
- Tabelle Hash o dizionari (Strutture che associano chiavi a valori)
- Set o insiemi (Un insieme di oggetti unici)

Python – le variabili

- Per definire una variabile si usa l’operatore =
- Ogni nome di variabile deve iniziare con una lettera o col carattere ‘_’ e può essere seguita da lettere, numeri e underscore
- Case-sensitive
- In Python è possibile anche l’assegnamento multiplo:

```
a, b, c = 2, 3, 5  # assegnamento multiplo
```

Python – le stringhe - indexing

- In Python, è possibile accedere agli elementi di una sequenza, usando la sintassi nomeSequenza[indice]. Questo restituirà l’elemento in posizione indice (il primo elemento ha sempre indice 0!)
- È inoltre possibile specificare indici negativi che partono dalla fine della sequenza (l’ultimo elemento ha indice -1, il penultimo -2, ecc.)
- >>> s = 'Python’

  >>> s[0] # elemento in posizione 0 (il primo) ‘P’

  >>> s[-1] # elemento in posizione -1 (l'ultimo) 'n'

Python – le stringhe - slicing

- La sintassi sequenza[inizio:fine] ci permette di ottenere una nuova sequenza dello stesso tipo che include tutti gli elementi partendo dall’indice inizio (incluso) all’indice fine (escluso)
- >>> s[0:2] # sottostringa con elementi da 0 (incluso) a 2 (escluso) 'Py’

  >>> s[:2]   # dall'inizio all'elemento con indice 2 (escluso) 'Py’

  >>> s[3:5] # dall'elemento con indice 3 (incluso) a 5(escluso) 'ho’

  >>> s[4:]   # dall'elemento con indice 4 (incluso) alla fine 'on’

  >>> s[-2:] # dall'elemento con indice -2 (incluso) alla fine 'on'

Python – le stringhe - sottostringhe

- Gli operatori **in** e **not in** possono essere usati per verificare se un elemento fa parte di una sequenza o no
- >>> 'P' in s # il carattere 'P' è contenuto nella stringa s, quindi torna True
- >>> 'x' in s # il carattere 'x' non è in s, quindi ritorna False
- >>> 'x' not in s # "not in" esegue l'operazione inversa, quindi torna True
- >>> 'Py' in s # la sottostringa 'Py' è contenuto nella stringa s, quindi torna True
- >>> 'py' in s # il controllo è case-sensitive, quindi ritorna False

Python – le stringhe

- CONCATENAZIONE: +
- RIPETIZIONE: *

```
>>> 'Ba' + 'na' * 2
'Banana’
```

- LUNGHEZZA: len()

```
>>> len(‘Banana’)
6
```

Python – le stringhe

- str.format() è un metodo che permette di inserire valori variabili in una stringa

```
>>> raggio = 8.4
>>> area = 3.14 * raggio**2
>>> circ = 2 * 3.14 * raggio
>>> s = "L'area e’ {}, la circonferenza e’ {}."
>>> s.format(area, circ)
"L'area e’ 221.5584, la circonferenza e’ 52.752."
```

Python – le tuple

- Una tupla è una sequenza immutabile di oggetti, in genere eterogenei

```
>>> t = 'abc', 123, 45.67  # la virgola crea la tupla
>>> t  # la rappresentazione di una tupla include sempre le ()
('abc', 123, 45.67)
>>> type(t)
<class 'tuple'>
>>> tp = ('abc', 123, 45.67)  # le () evitano ambiguità
>>> t == tp  # il risultato è equivalente, torna True
>>> len((1, 'a', 2.3))  # in questo caso le () sono necessarie, torna 3
```

Python – le tuple

- Supportano le operazioni comuni a tutte le sequenze, come indexing, slicing, contenimento, concatenazione, e ripetizione
- Sono immutabili: non posso riassegnare
- >>> t[0] = ‘pippo’ #errore
- È anche possibile usare funzioni e metodi come: len(), min(), max(), count(), ecc
- >>> t.count(‘c’) #torna il numero di occorrenze di ‘c’

Python – le liste

- Le liste vengono definite elencando tra parentesi quadre [ ] una serie di oggetti separati da virgole
- A differenza di tuple e stringhe che sono immutabili, le liste possono essere mutate

```
>>> nums = [0, 1, 2, 3]  # nuova lista di 4 elementi
>>> type(nums)  # verifichiamo che il tipo sia "list"
<class 'list'>
>>> empty = []  # nuova lista vuota
>>> one = ['Python']  # nuova lista con un elemento
>>> letters = ['a', 'b', 'c', 'd', 'e’]
```

Python – le liste

- >>> letters = ['a', 'b', 'c']
- >>> letters.append('d') # aggiunge 'd' alla fine ['a', 'b', 'c', 'd']
- >>> letters.extend(['e', 'f']) # aggiunge 'e' e 'f' alla fine ['a', 'b', 'c', 'd', 'e', 'f']
- >>> letters.append(['e', 'f']) # aggiunge la lista come elemento alla fine

  ['a', 'b', 'c', 'd', 'e', 'f', ['e', 'f']]
- >>> letters.pop() # rimuove e ritorna l'ultimo elemento (la lista) ['e', 'f’]
- >>> letters.pop(0) # rimuove e ritorna l'elemento in posizione 0 'a'
- >>> letters.remove('d') # rimuove l'elemento 'd'

Python – le liste

- >>> letters.reverse() # inverte l'ordine "sul posto" e non ritorna niente
- >>> letters # gli elementi sono ora in ordine inverso ['e', 'c', 'b']
- >>> letters[1] = 'x' # sostituisce l'elemento in posizione 1 ('c') con ‘x’
- >>> letters   # ['e', 'x', 'b']
- >>> del letters[1] # rimuove l'elemento in posizione 1 ('x')
- >>> letters   # ['e', 'b']
- >>> letters.clear() # rimuove tutti gli elementi rimasti
- >>> letters   # []

Python - dizionari

- I dizionari (dict) sono **mutabili** e **non ordinato** che contiene elementi (items) formati da una chiave (key) e un valore (value)
- Una volta che il dizionario è creato e valorizzato con un insieme di coppie <chiave, valore>, si può usare la chiave (che deve essere univoca) per ottenere il valore corrispondente
- DEFINIZIONE: elencando tra parentesi graffe { } una serie di elementi separati da virgole. Ogni elemento è una coppia ‘chiave’ : valore
- >>>d = {'a': 1, 'b': 2, 'c': 3} # nuovo dizionario di 3 elementi

Python - dizionari

- Le chiavi di un dizionario sono solitamente stringhe, ma è possibile usare anche altri tipi, a patto che sia possibile calcolarne l'hash e che siano consistenti (tutte con lo stesso tipo)
- I valori possono essere di qualsiasi tipo
- Per ottenere il valore associato a una chiave: dizionario['chiave']

  >>> d['a'] # ritorna il valore associato alla chiave 'a', cioè 1
- Per verificare se una chiave è presente nel dizionario si usa in o not in
- >>> 'b' in d # la chiave 'b' è presente, torna True
- >>> 'x' in d # la chiave 'x' non è presente in d, torna False

Python - dizionari

- Se viene specificata una chiave non presente nel dizionario: errore

  >>> d['x'] # se la chiave non esiste restituisce un KeyError

  Traceback (most recent call last): File "<stdin>", line 1, in <module>

  KeyError: ‘x’
- MODIFICA e INSERIMENTO: dizionario[‘chiave’] = nuovoValore

  >>> d['a'] = 10 # modifica il valore associato a una chiave esistente

  >>> d['x'] = 123 # crea un nuovo elemento, con chiave 'x' e valore 123
- CANCELLAZIONE: del dizionario[‘chiave’]

  >>> del d['x'] # rimuove l'elemento (chiave e valore) con chiave 'x'

Python – dizionari – metodi utili

- d.items() Restituisce gli elementi di d come un insieme di tuple
- d.keys() Restituisce le chiavi di d
- d.values() Restituisce i valori di d
- d.get(chiave, default) Restituisce il valore corrispondente a chiave se presente, altrimenti il valore di default (None se non specificato)
- d.pop(chiave, default) Rimuove e restituisce il valore corrispondente a chiave se presente, altrimenti il valore di default (dà KeyError se non specificato)
- d.popitem() Rimuove e restituisce un elemento arbitrario da d
- d.update(d2) Aggiunge gli elementi del dizionario d2 a quelli di d
- d.copy() Crea e restituisce una copia di d
- d.clear() Rimuove tutti gli elementi di d

Python - set

- I set (insiemi) vengono usati per rappresentare un insieme **non ordinato** di **oggetti unici;** eventuali duplicati vengono rimossi automaticamente
- DEFINIZIONE: tra parentesi graffe { } separati da virgole

  >>>nums = {10, 20, 30, 40} #nuovo set di 4 elementi

  >>> empty = set() # nuovo set vuoto
- E’ possibile creare un nuovo set contenente gli elementi univoci a partire da un altro oggetto

  >>> set('abracadabra') # trova l'insieme di lettere nella stringa
  {'d’, 'b', 'a', 'c', 'r’}

Python - set

- Supportano operazioni di base come: len(), min(), max(), in, not in, …
- INSERIMENTO: set.add(elem)

  ```
  >>> nums.add(50)  # aggiunge 50, nums = {40, 10, 20, 50, 30}
  ```
- COPIA: set.copy() , crea e restituisce una copia del set
- CANCELLAZIONE:
  - set.remove(elem) , errore se elem non presente
  - set.discard(elem) , rimuove elem se presente
  - set.pop() , rimuove e restituisce un elemento random dal set
  - set.clear() , rimuove tutti gli elementi dal set

Python – set – operazioni insiemistiche

Operatore	Metodo	Descrizione
	`s1.isdisjoint(s2)`	Restituisce `True` se i due set non hanno elementi in comune
`s1 <= s2`	`s1.issubset(s2)`	Restituisce `True` se `s1` è un sottoinsieme di `s2`
`s1 < s2`		Restituisce `True` se `s1` è un sottoinsieme *proprio* di `s2`
`s1 >= s2`	`s1.issuperset(s2)`	Restituisce `True` se `s2` è un sottoinsieme di `s1`
`s1 > s2`		Restituisce `True` se `s2` è un sottoinsieme *proprio* di `s1`
`s1 | s2 | ...`	`s1.union(s2, ...)`	Restituisce l'unione degli insiemi (tutti gli elementi)
`s1 & s2 & ...`	`s1.intersection(s2, ...)`	Restituisce l'intersezione degli insieme (elementi in comune a tutti i set)
`s1 - s2 - ...`	`s1.difference(s2, ...)`	Restituisce la differenza tra gli insiemi (elementi di `s1` che non sono negli altri set)
`s1 ^ s2`	`s1.symmetric_difference(s2)`	Restituisce gli elementi dei due set senza quelli comuni a entrambi
`s1 |= s2 | ...`	`s1.update(s2, ...)`	Aggiunge a `s1` gli elementi degli altri insiemi
`s1 &= s2 & ...`	`s1.intersection_update(s2, ...)`	Aggiorna `s1` in modo che contenga solo gli elementi comuni a tutti gli insiemi
`s1 -= s2 | ...`	`s1.difference_update(s2, ...)`	Rimuove da `s1` tutti gli elementi degli altri insiemi
`s1 ^= s2`	`s1.symmetric_difference_update(s2)`	Aggiorna `s1` in modo che contenga solo gli elementi non comuni ai due insiemi

Python – if elif else

- Le istruzioni condizionali vengono utilizzate quando vogliamo eseguire un blocco di codice solo nel caso in cui una condizione sia vera o falsa

```
if condizione :
        Istruzioni eseguite se la condizione è vera
elif condizione2:
            Istruzioni eseguite se la condizione2 è vera
else:
            Istruzioni eseguite se tutte le condizioni sono false
```

Python – if elif else - esempio

```
n = int(input('Inserisci un numero: ‘))
if n < 0:
    print(n, 'è negativo')
elif n > 0:
    print(n, 'è positivo')
else:
    print(n, 'è zero')
```

Python –ciclo for

- Ci permette di iterare su tutti gli elementi di un iterabile (liste, tuple, set, dizionari, ecc.) ed eseguire un determinato blocco di codice

  for *variabile* **in** *iterabile* :

  istruzioni

- Termina quando il blocco di codice (istruzioni) è stato eseguito per tutti i valori di *variabile* (quindi per tutti gli elementi di *iterabile)*

```
seq = [1, 2, 3, 4, 5]
for n in seq:
    print('Il quadrato di', n, 'è', n**2)
```

Python – range

- Spesso accade di voler lavorare su sequenze di numeri, Python fornisce una funzione, chiamata range, che permette di specificare uno valore iniziale o start (incluso), un valore finale o stop (escluso), e uno step, e che ritorna una sequenza di numeri interi

```
>>> range(5)  # ritorna un oggetto range con start uguale a 0 e stop uguale a 5
range(0, 5)
>>> list(range(5))  # convertendolo in lista possiamo vedere i valori
[0, 1, 2, 3, 4]
>>> list(range(5, 10))  # con 2 argomenti si può specificare lo start e lo stop
[5, 6, 7, 8, 9]
>>> list(range(0, 10, 2))  # con 3 argomenti si può specificare anche lo step
[0, 2, 4, 6, 8]
```

Python – ciclo while

- Il ciclo while itera fintanto che la condizione è vera

  While *condizione* :

  istruzioni

- Esempio:

```
>>> # rimuovi e stampa numeri da seq finché ne rimangono solo 3
>>> seq = [10, 20, 30, 40, 50, 60]
>>> while len(seq) > 3:
...     print(seq.pop())
```

Python – break e continue

- All’interno dei cicli possiamo utilizzare:
  - Break : interrompe il ciclo
  - Continue : interrompe l’iterazione corrente ed esegue quella successiva

```
>>> seq = ['alpha', 'beta', 'gamma', 'delta']
>>> for elem in seq:
...     if len(elem) == 4:
...         continue  # procedi all'elemento successivo
...     print('Sto controllando', elem)
...     if elem == 'gamma':
...         print('Elemento trovato!')
...         break  # elemento trovato, interrompi il ciclo
```

Python – comprehension

- Strumento che ci permette di creare in modo veloce e conveniente nuove liste, set, e dizionari partendo da una sequenza di valori esistenti
- Permettono anche di filtrare e trasformare gli elementi
- SINTASSI:
- Listcomp: [expr for elem in seq if cond]
- Setcomp: {expr for elem in seq}
- Dictcomp: {expr: expr for elem in seq}

Python – comprehension

- Per ogni elemento della sequenza l’espressione viene valutata e il risultato viene aggiunto alla list/set/dict
- Quanto tutti gli elementi sono stati creati, una nuova lista, set, o dizionario viene restituito

Python – comprehension

```
>>> [x**2 for x in range(10)] # list comprehension che crea una lista di quadrati
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> {x**3 for x in range(10)} # set comprehension che crea un set di cubi
{0, 1, 64, 512, 8, 343, 216, 729, 27, 125}

>>> {c: c.upper() for c in 'abcde’} # dict comprehension che mappa lettere lowercase all'equivalente uppercase
{'c': 'C', 'e': 'E', 'a': 'A', 'b': 'B', 'd': 'D'}
```

Python – comprehension

>>> [x for x in range(10) if x%2 == 0] # listcomp che filtra i numeri e prende solo quelli pari

[0, 2, 4, 6, 8]

>>> {c for c in 'aAbBcCdDeE' if c.isupper()} # setcomp che filtra le lettere e prende solo quelle uppercase

{'C', 'E', 'B', 'A', 'D’}

>>> [c+n for c in 'ABC' for n in '123’] # listcomp che crea tutte le combinazioni tra ABC e 123

['A1', 'A2', 'A3', 'B1', 'B2', 'B3', 'C1', 'C2', 'C3']

Virtual Environments - definizione

- Un virtual environment ("venv" per brevità) è una installazione isolata di Python
- Permettono di tenere sotto stretto controllo i pacchetti che servono all'esecuzione di un determinato progetto
- Concretamente, i venv sono delle directory con dentro una copia di Python e della libreria standard; con dentro tutti i pacchetti esterni che installerete via via, utili all’esecuzione di un determinato progetto

Virtual Environments - precisazione

- Il Virtual Environment è indipendente dai branch
- PyCharm ogni volta che cambi branch, ma in generale ogni volta che riscontra delle differenze tra il requirements.txt e il virtual environment, ti chiede di installare i nuovi requirements
- La prima volta che fai il checkout di un branch, se non trova una libreria necessaria, PyCharm ti chiede di installarlo (oppure si può sempre farlo a mano lanciando il comando)

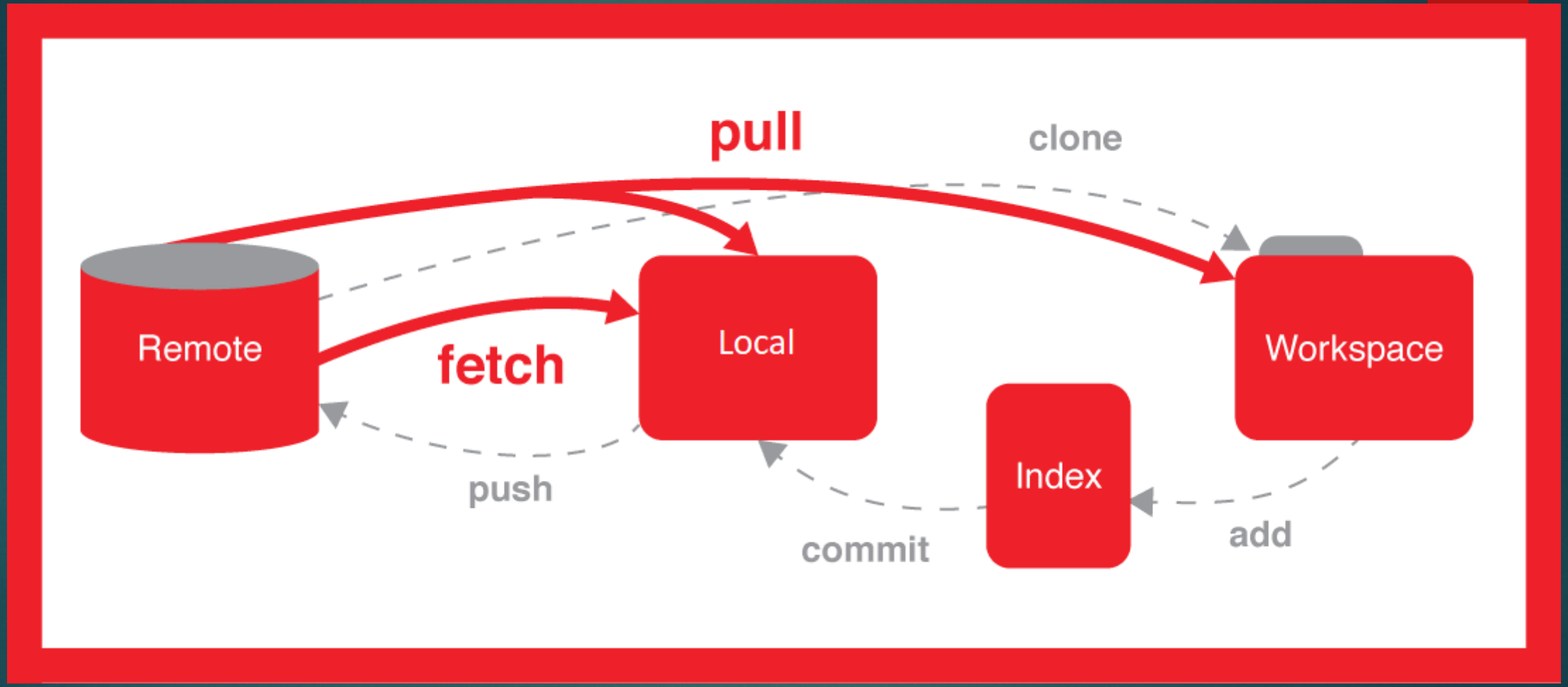
GitHub

- Per registrare il tuo account:

1. Apri le Impostazioni (Ctrl + Alt + S) e seleziona ‘Version Control | GitHub’ nel riquadro di sinistra.
2. Selezionare il tipo di autenticazione che si desidera utilizzare dall'elenco ‘Auth Type’:
   - Password: Se questa opzione è selezionata ed è abilitata l'autenticazione a due fattori nelle impostazioni dell'account GitHub, ti verrà chiesto di inserire un codice di autenticazione ogni volta che PyCharm ti richiede di accedere al tuo account GitHub.
   - Token: (raccomandato da GitHub per l'autenticazione da applicazioni di terze parti, poiché non richiede a PyCharm di ricordare la password).
3. Specificare le proprie credenziali in base al tipo di autenticazione selezionato e fare clic su OK.

GitHub – architettura

- REMOTE REPOSITORY: contiene il repository remoto ospitato su GitHub.com
- LOCAL REPOSITORY: contiene una copia del repository in locale (sulla tua macchina)
- WORKSPACE: contiene il codice modificato a partire dal LOCAL REPOSITORY


pull
clone
Remote
fetch
Local
Workspace
Index
push
commit
add

GitHub – comandi principali

- CREAZIONE DI UN NUOVO REPOSITORY:
  - Creare una nuova cartella, entrarci e eseguire il comando: git init
- CHECKOUT DI UN REPOSITORY GIA’ ESISTENTE:
  - Repository locale: git clone /pathDelRepository
  - Repository remoto: git clone gitRepositoryURL

GitHub – comandi principali

- AGGIUNGERE E VALIDARE
  - Dopo aver modificato un file, per aggiungere le modifiche (all'index): git add nomeFile

    git add *
  - Per validare le modifiche (aggiungerle nell'head e quindi nella copia in locale)

    git commit –m ''messaggio che spieghi le modifiche tra doppi apici''

GitHub – comandi principali

- SALVARE LE MODIFICHE NEL REPOSITORY REMOTO:

  git push origin nomeBranch

  Es.: git push origin master

- AGGIORNARE E INCORPORARE
  - Per aggiornare la propria copia locale: git pull
  - Il pull fa in maniera trasparente il ‘fetch’ (cioè recupera le modifiche) ed il ‘merge’ (cioè incorpora le modifiche alla tua versione)
  - A volte vi sono dei conflitti, si risolvono manualmente ed infine vanno aggiunte manualmente (git add nomeFileModificati)

GitHub – comandi principali

- SE QUALCOSA VA STORTO:
  - In caso di errore, si possono sostituire i cambiamenti fatti in locale con il comando:

    git ceckout – nomeFile (con l’ultimo contenuto presente in head)
  - Oppure

    git fetch origin + git reset --hard origin/master

    (per eliminare tutti i cambiamenti e i commit fatti in locale e recuperare l’ultima versione che c’era sul server)

GitHub

- Le esercitazioni verranno inserite durante il corso come nuovi branch nel repository WAAT-2018
- Per spostarsi in un branch:
  - Da terminale di PyCharm: git checkout nome_branch
  - In basso a destra Git: nome_branch. Aprendo il menu a tendina troviamo:
    - I ‘Local Branches’, ovvero i branch locali di cui si è già fatti il checkout, per cui occorre selezionare l’opzione ‘Checkout’
    - I ‘Remote Branches’, ovvero tutti quelli presenti nel repository di github remoto, per cui occorre selezionare ‘Checkout as new branch’

Pycharm

- Github:
  - Importare repository del corso (da GitHub) https://github.com/marcoortu/WAAT-2019
- Virtual Environment:
  - Aprire le Impostazioni (Ctrl + Alt + S) e selezionare ‘Project Interpreter’
  - Premere sull'ingranaggio a destra del campo e selezionare Add
  - New environment, assicurarsi in inserire la cartella corretta nel campo Location, scegliere l’interprete di Python 2 e premere OK

Ripasso Teoria

Architettura IR

- Per recuperare l’informazione, i sistemi di IR usano linguaggi di interrogazione basati su comandi testuali.
- Due concetti sono di fondamentale importanza, **query** ed **oggetto**:
  - Le **query** sono generalmente stringhe di parole-chiave rappresentanti l’informazione richiesta. Vengono digitate dall’utente in un sistema di IR
  - Un **oggetto** è un’entità che mantiene o racchiude informazioni in una banca dati. Un documento di testo, per esempio, è un oggetto di dati.
- A seguito di un’interrogazione, il sistema segnala il numero di documenti ritrovati e ordina i documenti per rilevanza

Passi di Pre-Elaborazione:

- Eliminazione di caratteri o di simboli di annotazione indesiderati (e.g. etichette, tag HTML, punteggiatura,...)
- Generazione delle sequenze di token basata su spazi.
- Stemming dei tokens in “radici” :

  computational -> comput
- Eliminazione delle parole irrilevanti (stopwords), e.g. un, il, esso/a, ...
- Costruzione del inverted index:

  keyword -> documenti che la contengono

Termine Indice

- I modelli classici considerano ogni documento come descritto da un insieme di parole chiave rappresentative dette anche termini indice
- Sia ki un termine indice, dj un documento e wij un peso associato alla coppia (ki,dj)
- Questo quantifica l’importanza del termine indice nel descrivere i contenuti semantici del documento: maggiore è il peso maggiormente il termine è significativo

Peso dei Termini

- Siano:
  - t il numero di termini indice nel sistema
  - ki un generico termine indice
  - K = {k1,...,kt} è l'insieme di tutti i termini indice
- Un peso wij>0 è associato a ogni termine indice ki di un documento dj
- Per un termine indice che non appare nel testo del documento, wij=0
- Al documento dj è associato un vettore di termini indice, rappresentato da:

  vec{dj} = (w1j,w2j,...,wtj)

Modelli di IR:

- **Modello Booleano:** i documenti sono rappresentati come insiemi (set) di parole chiave; pertanto diremo che il modello è di tipo set theoretic
- **Modello Vettoriale:** i documenti sono rappresentati come vettori in uno spazio t-dimensionale; pertanto diremo che il modello è algebrico
- **Modello Probabilistico:** il fondamento della modellazione della rappresentazione del documento è la teoria della probabilità; pertanto diremo che il modello è probabilistico

Modello booleano:

- Un documento è rappresentato tramite un insieme di *keyword (*parole chiave*)*
- Le interrogazioni sono espressioni booleane basate sulle keywords, connesse da operatori logici di tipo AND, OR e NOT:

  [[Rio & Brazil] | [Hilo & Hawaii]] & hotel & !Hilton]
- Output: L’insieme dei documenti definiti rilevanti
- Ogni decisione (appartenenza a tale insieme) e’ categoriale (e.g. binaria)
- Le variabili dei termini indice sono binarie, cioè wij appartengono a 0,1

Modello booleano:

- VANTAGGI:
  - Facile da comprendere
  - Facile da implementare (indici inversi)
- SVANTAGGI:
  - Poco flessibile: tutti (AND) o qualsiasi (OR)
  - Inadatto a query complesse

Esempio:

- Consideriamo la query: BRUTUS AND CAESAR
- Passi necessari per localizzare i documenti che rispondono alla query:
  - localizza BRUTUS nel dizionario
  - Ricerca la lista dei documenti che contengono BRUTUS
  - localizza CAESAR nel dizionario
  - Ricerca la lista dei documenti che contengono CAESAR
  - restituisci l’intersezione delle due liste all’utente

Vector Model

- Definisci tutti i termini trovati nei documenti. Vocabolario (V).
- Assegna i termini ai vettori ortogonali dello spazio.

  $|V| = N$
- Ogni termine *i-esimo* riceve un peso in un documento *j-esimo* (o query), *wij.*
- I documenti e le query sono vettori N-dimensionali:

  $dj = (w1j, w2j, \ldots, wNj)$

Vector Model

- Una collezione di n documenti viene rappresentata da una matrice di dimensione Nxn
  - N numero dei termini
  - n numero dei documenti
  - L’elemento wij di tale matrice esprime il peso del termine j-esimo nel documento i-esimo

Vector Model

- I modelli vettoriali si distinguono relativamente alle scelte di:
  - Pesatura dei temrini nelle interrogazioni
  - Pesatura dei temrini nei documenti
  - Metrica: Funzione di similarità
  - Criterio di accettazione

Pesatura: Frequenza

- Più frequenti i termini sono in un documento più importanti essi tendono ad essere (ad es. più significativi per il contenuto)

  f_{ij} = frequenza del termine *i* nel documento *j*

- Per normalizzare la frequenza (*tf*), *term frequency,* attraverso il corpus:

  $tf_{ij} = f_{ij} / max_k\{f_{kj}\}$

Pesatura: *Inverse Document Frequency*

- I termini che appaiono in molti documenti *differenti* sono *meno indicativi* del contenuto (es. *a, da, per, io, questo, ha, ho, è*...):

  df_i = *document frequency* del termine *i* = numero di docs che contengono il termine *i*

  id_{fi} = inverse document frequency del termine *i*, = *log2* (*N/ df i*)

  (con N: numero totale dei documenti)

Pesatura: TF-IDF

- Un indicatore che cattura le precedenti proprietà e' il fattore *tf-idf*:

  $w_{ij} = tf_{ij}\ idf_i = tf_{ij}\ log2\ (N/\ df_i)$

- Un termine che occorre frequentemente in un documento ma raramente nell'intera collezione riceve un peso alto

- Esistono molte varianti che forniscono alternative funzioni di pesatura. Sperimentalmente, *tf-idf* ha dimostrato ottime prestazioni

TF-IDF: Esempio

- Un documento ha i seguenti termini con le seguenti frequenze:

  A(3), B(2), C(1)

- Assumiamo una collezione di 10,000 docs in cui le frequenze globali dei termini sono:

  A(50), B(1300), C(250)

- Ne segue:

  A: *tf* = 3/3; *idf* = log(10000/50) = 5.3; *tf-idf*=5.3

  B: *tf* = 2/3; *idf* = log(10000/1300) = 2.0; *tf-idf*=1.3

  C: *tf* = 1/3; *idf* = log(10000/250) = 3.7; *tf-idf*=1.2

Metrica di Similarità

- Una metrica di similarità è una funzione che calcola il grado di similitudine tra due vettori

- Grazie all’uso di una metrica di similarità tra la query ed ogni documento è possibile:
  - ordinare i documenti dal più simile al meno simile
  - impostare una soglia al di sopra della quale respingere i documenti (per es. per controllare la numerosità dei risultati).

Misure di similarita’

- La similarita’ tra il documento ***dj*** e la query ***q*** puo’ essere calcolata secondo il *prodotto scalare*:

$$\text{sim}(d_j, q) = d_j \bullet q = \sum_{i=1}^{t} w_{ij} \cdot w_{iq}$$

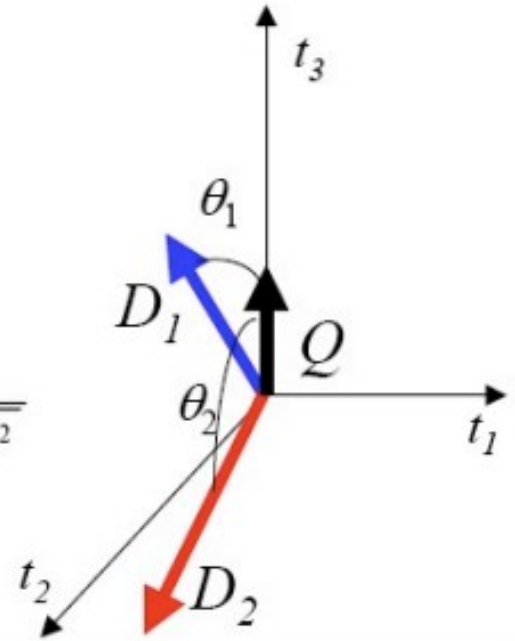
  dove *wij* è il peso del termine *i* nel documento *j* e *wiq* è il peso del termine *i* nella query

- Nel caso binario, il prodotto è la somma dei termini comuni tra query e documento (cardinalità della intersezione)
- Nel caso pesato, è la somma dei prodotti dei pesi dei soli termini in comune.

Coseno Similarità

- Misura il coseno dell'angolo tra due vettori.

$$\text{CosSim}(d_j, q) = \frac{\vec{d} \times \vec{q}}{|\vec{d}| \cdot |\vec{q}|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$



$D_1 = 2T_1 + 3T_2 + 5T_3$ $\quad \text{CosSim}(D_1, Q) = 10 / \sqrt{(4+9+25)(0+0+4)} = 0.81$

$D_2 = 3T_1 + 7T_2 + 1T_3$ $\quad \text{CosSim}(D_2, Q) = 2 / \sqrt{(9+49+1)(0+0+4)} = 0.13$

$Q = 0T_1 + 0T_2 + 2T_3$

D_1 e' 6 volte migliore di D_2 secondo la cosine similarity

Python: Esercizio1

- Scrivete 3 tuple che rappresentano 3 documenti, nel primo deve comparire Caesar, nel secondo Brutus nel terzo entrambi
- Scrivete una funzione booleanWeight:
  - Due parametri in input: term e document
  - Crea una collezione ‘words’ di tutte le parole di document. Che tipo di collezione?
  - Crea una collezione ‘terms’ con tutti i termini di words escludendo la punteggiatura e convertendoli in caratteri minuscoli. Che tipo di collezione?
  - Ritorna 1 se il termine ‘term’ è presente in ‘terms’, 0 altrimenti

Esercizio1: Soluzione

```
# coding=utf-8
from __future__ import division
import string
import numpy as np

doc1 = (
    "doc1",
    """una mattina, svegliandosi da sogni irrequieti Caesar
    si trovo nel suo letto trasformato in un insetto mostruoso
    """
)

doc2 = (
    "doc2",
    """Ho appena comprato un nuovo cane, lo ho chiamato Brutus
    """
)

doc3 = (
    "doc3",
    """Julio Caesar fu ucciso da suo figlio Brutus."""
)
```

Esercizio1: Soluzione

```
def binaryWeight(term, document):
    words = document[1].split()
    terms = set([word.translate(None, string.punctuation).lower() for word in words])
    return 1 if term in terms else 0
```

Esercizio1: Soluzione

```
if __name__ == '__main__':
    print(binaryWeight("caesar", doc1))
    print(binaryWeight("caesar", doc2))
    print(binaryWeight("caesar", doc3))
```

Python: Esercizio2

- Scrivete una funzione vectorize che:
  - Ha due parametri: ‘document’ e ‘documentsList’
  - Crea un set ordinato ‘terms’ con tutti i termini presenti in tutti i documenti (documentsList) sempre in minuscolo e senza punteggiatura
  - Stampa il set ordinato
  - Ritorna un ‘np.array’ che richiama la funzione ‘booleanWeight’ per tutti i termini in ‘terms’ e il documento ‘document’ passato come parametro

Esercizio 2: Soluzione

```
def vectorize(document, documentsList):
    terms = set([])
    for doc in documentsList:
        words = doc[1].split()
        words = set([word.translate(None, string.punctuation).lower() for word in words])
        terms.update(words)
    terms = sorted(terms)
    print(terms)
    return np.array([binaryWeight(term, document) for term in terms])
```

Esercizio 3 – vector model

- Converti tutti i documenti della collezione D in vettori dj pesati tramite tf-idf, per le keyword nel vocabolario V
- Converti l’interrogazione (query) in un vettore q pesato tramite tf-idf
- Per ogni dj ∈ D: Calcola sj=CosSim(dj,q)
- Ordina i documenti dj in ordine decrescente secondo sj e mostra i documenti ottenuti e ordinati all’utente

Riferimenti

- Teoria: Slides del Prof. Ortu:
  - Appendix - 01- Python Intro
  - Appendix - 02 – Git Intro
  - 01 - IR e Fondamenti di Web e Text Search 1
- Python: https://www.python.it
- Git: https://book.git-scm.com/doc