# CONTENUTI

Nel seguito vengono dettagliati i formati per le domande di tipo coderunner e essay. Viene inoltre definito il formato del file che serve per aggiungere informazioni di dettaglio sui settaggi del quiz

## Formato file info.quiz

In ogni directory è possibile inserire un file info.quiz che contiene una serie di direttive per impostare il quiz. Queste informazioni vengono considerate solo se il flag –canRedoQuiz è settato a False (in caso contrario, dal momento che suppongo che il backup sia per exercise non vengono lette le info in modo da NON settare delle datetime di inizio/fine del quiz). Il formato del file è quello dell’esempio seguente (le righe che iniziano con # sono di commento, e tutti i campi sono facoltativi):

**## open dd/mm/yy hh:mm --> starting date/time**

**open 12/12/2021 14:30**

**## close dd/mm/yy hh:mm --> closing date/time**

**close 12/12/2021 16:30**

**## duration minutes --> test length**

**duration 90**

**## delay minutes --> max opening delay allowed from open time WARNING: not working in import...**

**delay 25**

**## closeafter int --> time percentage from the beginning to allow closing the quiz**

**closeafter 40**

**## canredo true/false if users can redo... overrides basic settings (do this only in specific cases, otherwise keep the default)**

**canredo true**

## Formato file di testo delle domande coderunner

**QUESTION**

Identificatore della domanda (campo obbligatorio)

È la stringa che nel deposito domande di exercise permette di distinguere una domanda dall’altra. Questa stringa NON compare nel testo della domanda

***NOTA***: per permettere di rendere il programma più generale, e fare in modo che ogni domanda possa avere il suo punteggio) è possibile specificare nel campo question il voto della domanda, facendo finire il campo di testo libero con **@m mark** (dove **mark** è un valore intero o float)

Esempio (a questa domanda sarà associato un punteggio di 12):

**QUESTION**

**Esercizio di prova @m 12.0**

**TEXT**

Testo della domanda (campo obbligatorio). All’interno di moodle il testo deve essere in HTML. Per semplificare la vita, il testo della domanda deve essere **scritto in markdown** e viene convertito automaticamente in HTML (il markdown è decisamente più semplice da gestire che mettersi a scrivere tag HTML ovunque)

Link a definizione formato markdown:

<https://daringfireball.net/projects/markdown/syntax>

NOTA: per inserire codice nel markdown indentate ogni linea di codice con un tab. NON usate ‘’’ per delimitare un blocco di codice, dal momento che moodle poi non prende correttamente gli a capo (li considera come spazi).

NOTA2: i backtick (‘) in un testo normale creano un blocco a parte che interrompe la linea. Non si riesce quindi a inserire inline del codice (questo per come funziona moodle. Il tag <code> che viene generato, in moodle crea un box a parte su una linea separata interrompendo il testo)

NOTA3: nel markdown si può inserire codice HTML, che viene riportato as is. Per lo stesso motivo, non usate <> per definire parti del testo (ad esempio per identificare i campi di un file) perché poi vengono interpretati come tag HTML (e quindi non compaiono nel testo)

NOTA4: gli elenchi puntati o numerati in MD sono indicati con – item o 1 tem, 2 item. Però vengono poi visualizzati correttamente SOLO se sono preceduti da una linea vuota

**ANSWER**

Questo campo (facoltativo) contiene tutto quello che si vuole che compaia come indicazione/suggerimento nel testo della risposta. È un campo **plain text** perché deve essere **codice Python**. Si può anche non mettere niente. Essendo codice Python (e quindi compilabile) deve essere anche corretto (e.g., a volte qualcuno dimentica il ‘#’ prima dei commenti nella riga iniziale

***NOTA***: di deafult, la risposta viene popolata con un testo di default che permette di aprire e leggere i file (vedi la nota successiva). Nel caso NON si volesse approfittare di questa opzione di default (ma scrivere un proprio testo autonomo), basta mettere IN UNA QUALSIASI RIGA DI QUESTO CAMPO il testo:

# NO\_DEFAULT\_ANSWER

**FILES**

Nel caso l’esercizio preveda la lettura di file, si può fare in modo che i file siano già inglobati nella domanda stessa (anche se non vengono visualizzati dal momento che il “file browser” di coderunner non si può usare dal momento che la nostra versione di moodle non è aggiornata, e non si riesce ad aggiornare per tempo).

Si possono aggiungere tutti i file che si vogliono. In questo caso, basta elencare il nome (uno per riga). I file veri e propri si devono trovare nella working dir, devono essere file testuali (al momento…) e possono avere suffisso qualsiasi.

NOTA: visti i vari problemi con gli appelli passati e lettere non-ascii incluse nei file, salvate sempre i file in formato utf-8 (nel codice provo comunque a gestire eccezioni di formato e rileggere in nasi se ci sono problemi, quindi anche un file ANSI non da problemi).

NOTA: nel caso ci siano dei file in lettura, ho aggiunto in automatico al campo answer delle istruzioni automatiche di stampa del contenuto del file del tipo (notate il forcing del formato utf in lettura, sempre per le lettere accentate):

**print(open(filename,‘r’**, encoding="utf-8"**).read())**

Il campo è facoltativo. Inoltre, come scritto prima, nel caso NON si volesse approfittare di questa opzione di default (ma scrivere un proprio testo autonomo), basta mettere IN UNA QUALSIASI RIGA del campo ANSWER il testo:

# NO\_DEFAULT\_ANSWER

## Formato file di testo delle domande di tipo essay

**ESSAY**

Identificatore della domanda (campo obbligatorio)

È la stringa che nel deposito domande di exercise permette di distinguere una domanda dall’altra. Questa stringa NON compare nel testo della domanda

***NOTA***: per permettere di rendere il programma più generale, e fare in modo che ogni domanda possa avere il suo punteggio) è possibile specificare nel campo testo di ESSAY il voto della domanda, facendo finire il campo di testo libero con **@m mark** (dove **mark** è un valore intero o float)

Esempio (a questa domanda sarà associato un punteggio di 1):

**ESSAY**

**Esercizio di teoria @m 1**

**TEXT**

Testo della domanda (campo obbligatorio). Come nel caso precedente, anche qui va **scritto in markdown** (con le regole specificate sopra)

**ANSWER**

Questo campo (facoltativo) contiene tutto quello che si vuole che compaia come indicazione/suggerimento nel testo della risposta. Al contrario delle domande per coderunner, questo è un campo che va **scritto in markdown** (cosa che consente di avere più flessibilità nel formato)

## Esempio file di domande

**ESSAY**

**Somma ca2**

**TEXT**

**Qual è il risultato della seguente somma, nella rappresentazione in complemento a 2 su 8 bit?**

**10111110 + 01110011**

**ANSWER**

**Risultato:**

**QUESTION**

**Incroci**

**TEXT**

**# Incroci**

**Si scriva un programma per la gestione delle precedenze ad un incrocio di tre strade denominate N, S, E. Si scriva un programma per la gestione delle precedenze ad un incrocio di tre strade. Il programma deve leggere gli arrivi sulle varie file tramite un file che si chiama “arrivi.txt”. Ogni riga di tale file rappresenta un certo istante di tempo in questo modo:**

**TARGA\_N TARGA\_S TARGA\_E**

**Dove TARGA è una stringa di 7 caratteri, TARGA\_N indica un arrivo dalla strada N, TARGA\_S indica un arrivo dalla strada S, TARGA\_E indica arrivo dalla strada E. Nel caso in cui non ci sia una macchina proveniente da una determinata strada TARGA sarà rappresentata dalla stringa \*\*XXXXXX\*\*.**

**Per ogni istante di tempo considerato (ovvero per ogni riga del file in ingresso) il programma deve scrivere sul file di output “precedenze.txt”, la macchina che può attraversare l’incrocio secondo la seguente regola: ha precedenza la macchina della fila più lunga. Nel caso di file di ugual lunghezza si scelga la prima esaminata (quindi, l’ordine di precedenza è N->S->E). Terminata la lettura del file in input continuare il programma deve continuare a scrivere in ordine le targhe delle macchine che attraversano l’incrocio fino all’esaurimento delle stesse.**

**Una macchina che non passa ad un certo istante di tempo all’incrocio rimane in fila. I nuovi arrivi di macchine ad ogni istante di tempo si accodano alle rispettive file.**

**# Esempio file arrivi.txt (in input)**

**AA100BB CC200FF GG300HH**

**CC100ZZ YY200PP VV500KK**

**NN200KK XXXXXXX HH200DD**

**XXXXXXX FF600PP XXXXXXX**

**# Esempio esecuzione:**

**c:> python incrocio.py**

**File precedenze.txt (creato dal programma)**

**AA100BB**

**CC200FF**

**GG300HH**

**CC100ZZ**

**YY200PP**

**VV500KK**

**NN200KK**

**FF600PP**

**HH200DD**

**ANSWER**

**# this is an example of answer filling**

**# i file specificati NON sono visibili direttamente, ma esistono e si possono aprire**

**FILES**

**arrivi.txt**