ICDL
The Digital Skills Standard

AICA

ICDL PROFESSIONAL
COMPUTING

Syllabus 1.0

Versione italiana

Scopo

Questo documento presenta il syllabus di *ECDL – Computing*. Il syllabus descrive, attraverso i risultati del processo di apprendimento, la conoscenza e le capacità di un candidato. Il syllabus fornisce inoltre le basi per il test teorico e pratico relativo a questo modulo.

Nota del traduttore

La versione ufficiale in lingua inglese del Syllabus ECDL Computing Versione 1.01 è quella pubblicata sul sito web della Fondazione ECDL che si trova all'indirizzo **www.ecdl.org**. La presente versione italiana è stata tradotta a cura di AICA e rilasciata nel mese di giugno 2017.

Tanto la natura "definitoria" del testo, quanto la sua forma schematica costituiscono ostacoli di fronte ai quali è necessario trovare qualche compromesso; pur cercando di rendere al meglio in lingua italiana i concetti espressi nell'originale inglese, in alcuni casi sono evidenti i limiti derivanti dall'uso di un solo vocabolo per tradurre una parola inglese. Tale limite è particolarmente riduttivo per i verbi che dovrebbero identificare con maggiore esattezza i requisiti di conoscenza o competenza: moltissime voci contengono verbi come *understand*, *know*, *know about*, che sono stati solitamente tradotti con "comprendere", "conoscere", "sapere", ma che potrebbero valere anche per "capire", "intendere", "definire", "riconoscere", "essere a conoscenza" ...
Per alcuni vocaboli tecnici è inoltre invalso nella lingua l'uso del termine inglese (es. *hardware*, *software*), e in molti casi – pur cercando di non assecondare oltre misura questa tendenza – si è ritenuto più efficace attenersi al vocabolo originale o riportarlo tra parentesi per maggior chiarezza.
Si invitano i lettori che abbiano particolari esigenze di analisi approfondita dei contenuti a fare riferimento anche alla versione inglese di cui si è detto sopra.

Limitazione di responsabilità

Benché la Fondazione ECDL abbia messo ogni cura nella preparazione di questa pubblicazione, la Fondazione ECDL non fornisce alcuna garanzia come editore riguardo la completezza delle informazioni contenute, né potrà essere considerata responsabile per eventuali errori, omissioni, inaccuratezze, perdite o danni eventualmente arrecati a causa di tali informazioni, ovvero istruzioni ovvero consigli contenuti nella pubblicazione. Le informazioni contenute in questa pubblicazione non possono essere riprodotte né nella loro interezza né parzialmente senza il permesso e il riconoscimento ufficiale da parte della Fondazione ECDL. La Fondazione ECDL può effettuare modifiche a propria discrezione e in qualsiasi momento senza darne notifica.

[1] Tutti i riferimenti alla Fondazione ECDL riguardano la European Computer Driving Licence Foundation Limited.

Computing

Il presente modulo *ECDL – Computing* definisce i concetti e le competenze fondamentali necessari all'utilizzo del pensiero computazionale e alla programmazione per la creazione di semplici programmi per computer.

Obiettivi del modulo

Il candidato che ha superato il test è in grado di:

- Comprendere i concetti fondamentali relativi all'informatica e alle attività tipiche necessarie alla creazione di un programma.
- Comprendere e usare tecniche di pensiero computazionale quali decomposizione del problema, riconoscimento di modelli, astrazione e algoritmi per analizzare un problema e sviluppare soluzioni.
- Scrivere, verificare e modificare algoritmi per un programma usando schemi di flusso e pseudocodice.
- Comprendere i principi e i termini fondamentali associati alla programmazione e l'importanza di codice ben strutturato e documentato.
- Comprendere e usare costrutti di programmazione all'interno di un programma, quali variabili, tipi di dati e logica.
- Migliorare l'efficienza e la funzionalità utilizzando iterazioni, istruzioni condizionali, procedure e funzioni all'interno di un programma, insieme ad eventi e comandi.
- Sottoporre un programma a test e debug, assicurandosi che risponda ai requisiti richiesti prima del rilascio.

SEZIONE		TEMA		RIF.	Argomento
1	**Termini informatici**	*1.1*	*Concetti fondamentali*	1.1.1	Definire l'espressione "elaborazione dati".
				1.1.2	Definire l'espressione "pensiero computazionale".
				1.1.3	Definire il termine programma.
				1.1.4	Definire il termine codice. Distinguere tra codice sorgente e codice macchina.
				1.1.5	Comprendere le espressioni "descrizione del programma" e "specifiche del programma".
				1.1.6	Riconoscere le attività tipiche nella creazione di un programma: analisi, progettazione, programmazione, test, miglioramento.
				1.1.7	Comprendere la differenza tra linguaggio formale e linguaggio naturale.
2	**Metodi di pensiero computazionale**	*2.1*	*Analisi del problema*	2.1.1	Delineare i metodi tipici usati nel pensiero computazionale: decomposizione, riconoscimento di modelli, astrazione, algoritmi.
				2.1.2	Usare la decomposizione del problema per ridurre dati, processi o un problema complesso in parti più piccole.
				2.1.3	Identificare modelli all'interno di problemi piccoli e decomposti.
				2.1.4	Usare l'astrazione per filtrare dettagli non necessari durante l'analisi di un problema.

SEZIONE		TEMA		RIF.	Argomento
				2.1.5	Comprendere come vengono usati gli algoritmi nel pensiero computazionale.
		2.2	*Algoritmi*	2.2.1	Definire il termine “sequenza” dei costrutti di programmazione. Illustrare lo scopo delle sequenze nella progettazione di algoritmi.
				2.2.2	Riconoscere i possibili metodi di rappresentazione del problema quali schemi di flusso, pseudocodice.
				2.2.3	Riconoscere i simboli degli schemi di flusso, quali avvio/stop, elaborazione, decisione, ingresso/uscita, connettore, freccia.
				2.2.4	Illustrare la sequenza delle operazioni rappresentate da uno schema di flusso o da pseudocodice.
				2.2.5	Scrivere un algoritmo accurato sulla base di una descrizione usando una tecnica quale schema di flusso o pseudocodice.
				2.2.6	Correggere errori in un algoritmo, quali elementi mancanti in un programma, sequenza non corretta, risultato non corretto di una decisione.
3	**Iniziare a programmare**	*3.1*	*Per iniziare*	3.1.1	Descrivere le caratteristiche di codice ben strutturato e documentato, quali indentazione, commenti appropriati, nomi descrittivi.
				3.1.2	Usare semplici operatori aritmetici per eseguire calcoli in un programma: +, -, /, *.
				3.1.3	Comprendere la precedenza degli operatori e l’ordine di valutazione in espressioni complesse. Comprendere come usare le parentesi per strutturare espressioni complesse.
				3.1.4	Comprendere il termine parametro. Illustrare lo scopo dei parametri in un programma.
				3.1.5	Definire il termine “commento” dei costrutti di programmazione. Illustrare lo scopo di un commento in un programma.
				3.1.6	Usare i commenti in un programma.
		3.2	*Variabili e tipi di dati*	3.2.1	Definire il termine “variabile” dei costrutti di programmazione. Illustrare lo scopo di una variabile in un programma.
				3.2.2	Definire e inizializzare una variabile.
				3.2.3	Assegnare un valore a una variabile.
				3.2.4	Usare in un programma variabili con nomi appropriati per calcoli e memorizzazione di valori.
				3.2.5	Usare tipi di dati in un programma: stringa, carattere, intero, virgola mobile, booleano.
				3.2.6	Usare un tipo di dati aggregato in un programma, quali vettore, lista, tupla.

SEZIONE		TEMA		RIF.	Argomento
				3.2.7	Usare dati inseriti da un utente in un programma.
				3.2.8	Inviare dati verso uno schermo in un programma.
4	**Costruire con il codice**	*4.1*	*Logica*	4.1.1	Definire l'espressione "test logico" dei costrutti di programmazione. Illustrare lo scopo di un test logico in un programma.
				4.1.2	Riconoscere i tipi di espressioni in logica booleana che generano un valore vero o falso, quali =, >, <, >=, <=, <>, !=, ==, AND, OR, NOT.
				4.1.3	Usare espressioni di logica booleana in un programma.
		4.2	*Iterazione*	4.2.1	Definire il termine "ciclo" dei costrutti di programmazione. Illustrare lo scopo e i vantaggi dei cicli in un programma.
				4.2.2	Riconoscere i tipi di cicli usati per le iterazioni: for, while, repeat.
				4.2.3	Usare l'iterazione (cicli) in un programma, quali for, while, repeat.
				4.2.4	Comprendere il termine "ciclo infinito".
				4.2.5	Comprendere il termine "ricorsione".
		4.3	*Condizionalità*	4.3.1	Definire il termine "istruzione condizionale" dei costrutti di programmazione. Illustrare lo scopo delle istruzioni condizionali in un programma.
				4.3.2	Usare le istruzioni condizionali IF…THEN…ELSE in un programma.
		4.4	*Procedure e funzioni*	4.4.1	Comprendere il termine "procedura". Illustrare lo scopo di una procedura in un programma.
				4.4.2	Scrivere e assegnare un nome a una procedura in un programma.
				4.4.3	Comprendere il termine "funzione". Illustrare lo scopo di una funzione in un programma.
				4.4.4	Scrivere e assegnare un nome a una funzione in un programma.
		4.5	*Eventi e comandi*	4.5.1	Comprendere il termine "evento". Illustrare lo scopo di un evento in un programma.
				4.5.2	Usare i gestori degli eventi ("handler"), quali clic del mouse, input da tastiera, clic su pulsante, timer.
				4.5.3	Usare librerie generiche disponibili, quali matematiche, random, data/ora.
5	**Test, debug e rilascio**	*5.1*	*Esecuzione, test e debug*	5.1.1	Comprendere i vantaggi di sottoporre a test e debug un programma per risolvere gli errori.
				5.1.2	Comprendere i tipi di errori in un programma, quali sintattici e logici.
				5.1.3	Eseguire un programma.

SEZIONE	TEMA	RIF.	Argomento
		5.1.4	Identificare e correggere un errore sintattico in un programma, quale errore di ortografia, punteggiatura mancante.
		5.1.5	Identificare e correggere un errore logico in un programma, quale espressione booleana non corretta, tipo di dato non corretto.
	5.2 Rilascio	5.2.1	Controllare che il programma rispetti i requisiti della descrizione iniziale.
		5.2.2	Descrivere il programma completato, comunicando scopo e valore.
		5.2.3	Individuare arricchimenti e miglioramenti al programma che potrebbero soddisfare ulteriori esigenze.