ICDL
The Digital Skills Standard

ICDL Professional

COMPUTING

Syllabus 1.0

Syllabus

ICDL Germany

Zweck

Dieses Dokument beschreibt den Lehrplan für das ICDL Modul Computing. Der Lehrplan beschreibt anhand der Lernziele die Kenntnisse und Fähigkeiten, die ein Kandidat für dieses Modul besitzen sollte. Der Lehrplan bildet auch die Grundlage für den theoretischen und praktischen Test zu diesem Modul.

Disclaimer

Obwohl bei der Erstellung dieser Publikation alle Sorgfalt aufgewendet wurde, übernimmt die ICDL Foundation als Herausgeber der englischen Originalversion keine Gewähr für die Vollständigkeit der darin enthaltenen Informationen. Weiterhin übernimmt die ICDL Foundation keine Verantwortung oder Haftung für etwaige Fehler, Auslassungen, Ungenauigkeiten, Verluste oder Schäden, die aufgrund von Informationen, Anweisungen oder Ratschlägen in dieser Veröffentlichung entstehen. Änderungen können von der ICDL Foundation nach eigenem Ermessen und jederzeit ohne vorherige Ankündigung vorgenommen werden.

DLGI
Dienstleistungsgesellschaft für Informatik
Am Bonner Bogen 6
53227 Bonn
Tel.: 0228- 688-448-0 Fax: 0228- 688-448-99

E-Mail: info@dlgi.de
URL: www.dlgi.de
URL: www.icdl.de

Computing

Dieses Modul behandelt die wesentlichen Kenntnisse und Fertigkeiten, die für "informatisches Denken" und Programmieren notwendig sind, um kleine einfache Computerprogramme zu entwickeln.

Modulziele

Kandidatinnen und Kandidaten sollen:

- Die grundlegenden Konzepte von *Computing* und typische Aktivitäten im Zusammenhang mit Programmierung verstehen.
- Informatisches Denken (*Computational Thinking*) wie Problemzerlegung, Muster- bzw. Strukturerkennung, Abstraktion und Algorithmen verstehen bzw. anwenden, um ein Problem zu analysieren und entsprechende Lösungen zu entwickeln.
- Algorithmen für ein Programm unter Verwendung von Flussdiagrammen und Pseudocodes schreiben, testen und verbessern.
- Die wichtigsten Grundsätze und Begriffe im Zusammenhang mit Programmierung, und die Relevanz von gut strukturiertem und dokumentiertem Code verstehen.
- Programmierkonstrukte wie Variablen, Datentypen und Logik in einem Programm verstehen und verwenden.
- Effizienz und Funktionalität durch Verwendung von Iteration (mehrfache Zuweisung), bedingte Anweisungen, Prozeduren und Funktionen, sowie Ereignisse und Befehle in einem Programm verbessern.
- Ein Programm vor der Freigabe testen und debuggen, sowie sicherstellen, dass es alle Anforderungen erfüllt.

Kategorie	Wissensgebiet	Nr.	Lernziel
1 Computing Begriffe	*1.1 Grundlegende Begriffe*	1.1.1	Den Begriff *Computing* definieren.
		1.1.2	Den Begriff *Computational Thinking* definieren.
		1.1.3	Den Begriff Programm definieren.
		1.1.4	Den Begriff Code definieren. Zwischen Quellcode und Maschinencode unterscheiden.
		1.1.5	Die Begriffe Programmbeschreibung und Programmspezifizierung und deren Unterschied verstehen.
		1.1.6	Typische Aktivitäten zur Programmentwicklung kennen: Analyse, Design, Programmierung, Testen, Verbesserung.
		1.1.7	Den Unterscheid zwischen formaler und natürlicher Sprache verstehen.
2 Methoden des Computational Thinking	*2.1 Problemanalysen*	2.1.1	Typische Methoden des *Computational Thinking* aufzeigen können: Dekomposition, Muster erkennen, Abstraktion, Algorithmus.

Kategorie	Wissensgebiet	Nr.	Lernziel
		2.1.2	Problemdekomposition verwenden, um Daten, Prozesse oder komplexe Probleme in kleinere Einzelteile zu zerlegen.
		2.1.3	Muster in kleineren, zerlegten Problemen erkennen.
		2.1.4	Abstraktion anwenden, um unwichtige Details bei der Problemanalyse herauszufiltern.
		2.1.5	Verstehen, wie Algorithmen im *Computational Thinking* verwendet werden.
	2.2 Algorithmen	2.2.1	Den Begriff Sequenz beim Programmplanungskonstrukt definieren. Sinn und Zweck der Sequenzierung beim Entwurf von Algorithmen darlegen.
		2.2.2	Mögliche Methoden zur Problempräsentation kennen, wie: Ablaufdiagramme, Pseudocode.
		2.2.3	Symbole von Ablaufdiagrammen kennen, wie: Start/Stop, Prozess, Entscheidung, Input/Output, Connector, Pfeil.
		2.2.4	Die Sequenz einzelner Operationen in einem Ablaufdiagramm oder Pseudocode darlegen.
		2.2.5	Einen fehlerfreien Algorithmus auf Basis einer Beschreibung unter Verwendung von Ablaufdiagramm und/oder Pseudocode schreiben.
		2.2.6	Fehler in Algorithmen beheben, wie: fehlende Programmelemente, falsche Sequenzen, falsches Entscheidungsergebnis.
3 Programmieren	*3.1 Grundlagen*	3.1.1	Eigenschaften eines gut strukturierten und dokumentierten Codes beschreiben, wie: Einrückung, richtige Kommentare, aussagekräftige Bezeichnung.
		3.1.2	Einfache arithmetische Operatoren für Berechnungen im Programm kennen: +, -, /, *.
		3.1.3	Den Vorrang von Operatoren und die Reihenfolge von Evaluierung in komplexen Ausdrücken verstehen. Die Verwendung von Klammern bei komplexen Ausdrücken verstehen.

Kategorie	Wissensgebiet	Nr.	Lernziel
		3.1.4	Den Begriff Parameter verstehen. Sinn und Zweck von Parametern in einem Programm darlegen.
		3.1.5	Den Begriff Kommentar in der Programmierung definieren. Sinn und Zweck eines Kommentars innerhalb eines Programms darlegen.
		3.1.6	Kommentare in einem Programm verwenden.
	3.2 Variablen und Datentypen	3.2.1	Den Begriff Variable in einem Programm definieren. Sinn und Zweck einer Variablen innerhalb eines Programms darlegen.
		3.2.2	Eine Variable definieren und initialisieren.
		3.2.3	Einer Variablen einen Wert zuordnen.
		3.2.4	Korrekt benannte Variablen zur Berechnung und Speicherung von Werten in Programmen verwenden.
		3.2.5	Datentypen in Programmen verwenden: String (Zeichenkette), Zeichen, integer, float, Boolean.
		3.2.6	Aggregierte Datentypen im Programm verwenden, wie: Array, Liste, Tuple.
		3.2.7	Daten-Input eines Nutzers in einem Programm verwenden.
		3.2.8	Daten-Output auf einem Monitor in einem Programm verwenden.
4 Code verwenden	*4.1 Logisch*	4.1.1	Den Begriff *Logic Test* in einem Programm definieren. Sinn und Zweck eines *Logig Tests* in einem Programm darlegen.
		4.1.2	Verschiedene Arten Boolscher Ausdrücke kennen, um einen richtigen oder falschen Wert zu erzeugen, wie: =, >, <, >=, <=, <>, !=, ==, AND, OR, NOT.
		4.1.3	Boolsche Ausdrücke in einem Programm verwenden.
	4.2 Durchlauf	4.2.1	Den Begriff Schleife in einem Programm-Konstrukt definieren. Sinn und Vorzug einer Schleife in einem Programm darlegen.

Kategorie	Wissensgebiet	Nr.	Lernziel
		4.2.2	Verschiedene Arten von Schleifen für den Durchlauf kennen: *For-Schleife, While-Do-Schleife, Repeat-Schleife*.
		4.2.3	Schleifen bei einem Programm verwenden, wie: *for, while, repeat*.
		4.2.4	Den Begriff Endlosschleife (*Infinite Loop*) verstehen.
		4.2.5	Den Begriff Rekursion verstehen.
	4.3 Bedingtheit	4.3.1	Den Begriff bedingte Anweisung im Programm verstehen. Sinn und Zweck einer bedingten Anweisung in einem Programm darlegen.
		4.3.2	Bedingte Anweisungen IF…THEN...ELSE in einem Programm verwenden.
	4.4 Prozeduren und Funktionen	4.4.1	Den Begriff Prozedur verstehen. Sinn und Zweck einer Prozedur in einem Programm darlegen.
		4.4.2	Eine Prozedur in einem Programm schreiben und benennen.
		4.4.3	Den Begriff Funktion verstehen. Sinn und Zweck einer Funktion in einem Programm darlegen.
		4.4.4	Eine Funktion in einem Programm schreiben und benennen.
	4.5 Ereignisse und Kommandos	4.5.1	Den Begriff Ereignis verstehen. Sinn und Zweck eines Ereignisses darlegen.
		4.5.2	Event-Handler verwenden, wie: Mouseklick, Tastatureingabe, Klick auf Button, Timer.
		4.5.3	Generische Bibliotheken verwenden, wie: *math, random, time.*
5 Test, Debug und Release	*5.1 Run, Test und Debug*	5.1.1	Verstehen, warum man ein Programm Testen und Debuggen sollte, um Fehler zu beheben.
		5.1.2	Verschiedene Fehlerarten in einem Programm verstehen, wie: Syntax, Logik.
		5.1.3	Ein Programm ausführen.

Kategorie	Wissensgebiet	Nr.	Lernziel
		5.1.4	Syntaxfehler in einem Programm erkennen und beheben, wie: falsche Schreibweise, fehlende Interpunktion.
		5.1.5	Logische Fehler in einem Programm erkennen und beheben, wie: falscher Boolscher Ausdruck, falscher Datentyp.
	5.2 Release	5.2.1	Das Programm mit den ursprünglich beschriebenen Anforderungen abgleichen.
		5.2.2	Das fertig gestellte Programm beschreiben, Sinn und Zweck kommunizieren.
		5.2.3	Optimierungsmöglichkeiten, Verbesserungen am Programm erkennen, Verbesserungen am Programm, die zusätzliche, in Zusammenhang stehende Bedürfnisse decken.