Privater Masterstudiengang
Informatik und Programmiersprachen

tech technologische universität

tech technologische universität

Privater Masterstudiengang
Informatik und Programmiersprachen

- Modalität: **online**
- Dauer: **12 Monate**
- Qualifizierung: **TECH Technologische Universität**
- Zeitplan: **in Ihrem eigenen Tempo**
- Prüfungen: **online**

Internetzugang: www.techtitute.com/de/informatik/masterstudiengang/masterstudiengang-informatik-programmiersprachen

Index

01

Präsentation

IT-Experten müssen ihre Kenntnisse auf dem neuesten Stand halten, um in ihrem Fachgebiet weiterhin optimal arbeiten zu können, ohne die Fortschritte zu verpassen, die in diesem Bereich in schwindelerregendem Tempo gemacht werden. Dieses Update soll den Studenten ein vollständiges und vertieftes Wissen über die wesentlichen Kenntnisse und die interessantesten Neuerungen beim Entwurf von Algorithmen für die Entwicklung von IT-Projekten vermitteln, wobei die innovativsten und effizientesten Methoden des Sektors eingesetzt werden.

Erwerben Sie in einem hochkompetenten Masterstudiengang die grundlegenden Kenntnisse der Informatik und deren erfolgreiche Anwendung bei der Entwicklung von IT-Projekten"

Dieses Programm konzentriert sich auf die Grundlagen der Programmierung und Datenstruktur, Algorithmen und Komplexität, sowie auf den fortgeschrittenen Entwurf von Algorithmen, fortgeschrittene Programmierung, Sprachprozessoren und Computergrafik, neben anderen Aspekten, die mit diesem Bereich der Informatik zusammenhängen.

Im Rahmen dieses Programms erhalten die Studierenden spezifische Instrumente und Fähigkeiten, um ihre berufliche Tätigkeit im breiten Umfeld der Informatik und der Sprachen erfolgreich auszubauen. Es geht um Schlüsselkompetenzen wie die Kenntnis der Realität und der täglichen Praxis in verschiedenen IT-Bereichen und um die Entwicklung von Verantwortung bei der Überwachung und Beaufsichtigung ihrer Arbeit sowie von spezifischen Fähigkeiten in diesem Bereich.

Da es sich um ein 100%iges Online-Programm handelt, ist der Student nicht an feste Stundenpläne oder die Notwendigkeit gebunden, sich an einen anderen physischen Ort zu begeben, sondern kann zu jeder Tageszeit auf die Inhalte zugreifen und so sein Arbeits- oder Privatleben mit seinem akademischen Leben in Einklang bringen.

Das Dozententeam dieses Programms in Informatik und Sprachen hat eine sorgfältige Auswahl der einzelnen Themen dieses Updates getroffen, um den Studenten eine möglichst umfassende Studienmöglichkeit zu bieten, die immer mit dem aktuellen Zeitgeschehen verbunden ist.

Dieser **Privater Masterstudiengang in Informatik und Programmiersprachen** enthält das vollständigste und aktuellste Programm auf dem Markt. Die hervorstechendsten Merkmale sind:

- Die Ausarbeitung von Fallstudien, die von Experten für Informatik und Programmiersprachen vorgestellt werden
- Der anschauliche, schematische und äußerst praxisnahe Inhalt soll wissenschaftliche und praktische Informationen zu den für die berufliche Praxis wesentlichen Disziplinen vermitteln
- Die praktischen Übungen, bei denen der Selbstbewertungsprozess zur Verbesserung des Lernens durchgeführt werden kann
- Sein besonderer Schwerpunkt liegt auf innovativen Methoden in den Bereichen Informatik und Programmiersprachen
- Theoretische Vorträge, Fragen an den Experten, Diskussionsforen zu kontroversen Themen und individuelle Reflexionsarbeit
- Die Verfügbarkeit des Zugangs zu Inhalten von jedem festen oder tragbaren Gerät mit Internetanschluss

Eine außergewöhnliche Gelegenheit, auf bequeme und einfache Weise die mathematischen und grundlegenden Verfahren und Kenntnisse zu erlernen, die für eine qualitativ hochwertige Computerprogrammierung erforderlich sind"

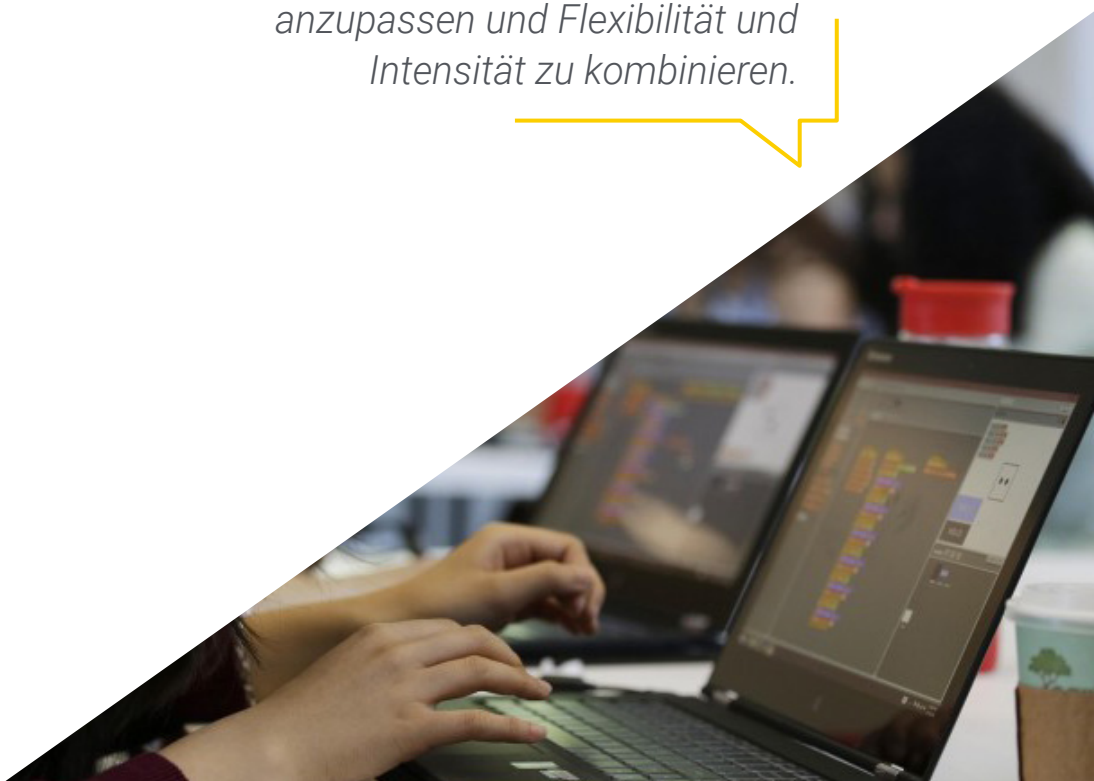
Ein privater Masterstudiengang, der sich auf die auf dem Markt am meisten geschätzte Bildungstechnologie stützt, mit audiovisuellen und Lernsystemen, die es Ihnen ermöglichen, schneller und bequemer zu lernen"

Die multimedialen Inhalte, die mit den neuesten Bildungstechnologien entwickelt wurden, ermöglichen den Fachleuten ein situiertes und kontextbezogenes Lernen, d. h. eine simulierte Umgebung, die eine immersive Aktualisierung ermöglicht, die auf die Ausbildung in realen Situationen programmiert ist.

Das Konzept dieses Studiengangs konzentriert sich auf problemorientiertes Lernen, bei dem die Fachkraft versuchen muss, die verschiedenen Situationen aus der beruflichen Praxis zu lösen, die während des gesamten Studiengangs gestellt werden. Zu diesem Zweck wird der Fachkraft ein innovatives interaktives Videosystem zur Verfügung gestellt, das von renommierten und erfahrenen Computer- und Sprachexperten entwickelt wurde.

Wir stellen Ihnen ein umfangreiches und übersichtliches didaktisches Material zur Verfügung, das alle aktuellen Themen umfasst, die für Fachleute von Interesse sind, die sich im Bereich Informatik und Sprachen weiterentwickeln wollen.

Ein Studium mit hoher pädagogischer Wirkung, das es Ihnen ermöglicht, den Aufwand an Ihre Bedürfnisse anzupassen und Flexibilität und Intensität zu kombinieren.



02

Ziele

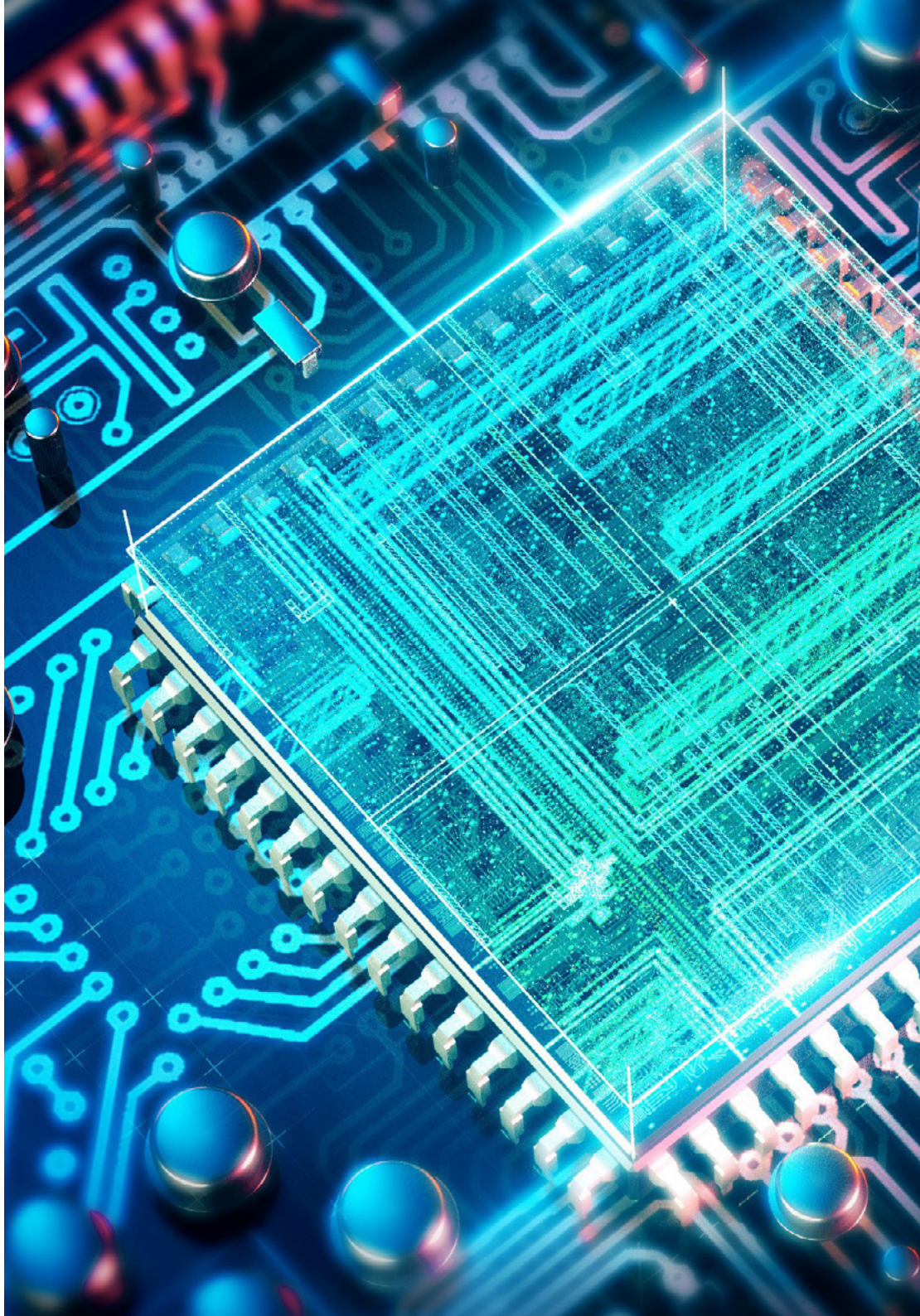
Der Studiengang Informatik und Sprachen wurde speziell für Berufstätige entwickelt, die in diesem Bereich schnell und mit echter Qualität vorankommen wollen. Er ist auf der Grundlage realistischer und hochwertiger Ziele organisiert, die ihn oder sie auf eine andere Ebene der Arbeit in diesem Bereich bringen werden.

Unser Ziel ist es, den Fachleuten im Bereich der Informatik ein qualitativ hochwertiges Update zu bieten, das es ihnen ermöglicht, mit Kompetenz im Bereich Informatik und Programmiersprachen zu intervenieren“

Allgemeines Ziel

- Wissenschaftliche und technologische Fortbildung sowie Vorbereitung auf die berufliche Praxis in den Bereichen Informatik und Programmiersprachen, und zwar mit einer transversalen und vielseitigen Aktualisierung, die an die neuen Technologien und Innovationen in diesem Bereich angepasst ist

Nutzen Sie die Gelegenheit und machen Sie den Schritt, sich über die neuesten Entwicklungen im Bereich Informatik und Programmiersprachen zu informieren"



Spezifische Ziele

Modul 1. Grundlagen der Programmierung

- Verständnis der grundlegenden Struktur eines Computers, der Software und allgemeinen Programmiersprachen
- Algorithmen entwerfen und interpretieren lernen, die die notwendige Grundlage für die Entwicklung von Computerprogrammen sind
- Die wesentlichen Elemente eines Computerprogramms verstehen, wie z.B. die verschiedenen Datentypen, Operatoren, Ausdrücke, Anweisungen, E/A- und Steueranweisungen
- Verstehen der verschiedenen Datenstrukturen, die in allgemeinen Programmiersprachen zur Verfügung stehen, sowohl statisch als auch dynamisch, und Erwerb der wesentlichen Kenntnisse für den Umgang mit Dateien
- Die verschiedenen Softwaretesttechniken und die Bedeutung der Erstellung einer guten Dokumentation zusammen mit einem guten Quellcode verstehen
- Erlernen der grundlegenden Konzepte der Programmiersprache C++, einer der am häufigsten verwendeten Sprachen der Welt

Modul 2. Datenstruktur

- Die Grundlagen der Programmierung in der Sprache C++, einschließlich Klassen, Variablen, bedingte Ausdrücke und Objekte
- Abstrakte Datentypen, lineare Datenstrukturtypen, einfache und komplexe hierarchische Datenstrukturen und deren Implementierung in C++ verstehen
- Die Funktionsweise von fortgeschrittenen Datenstrukturen, die nicht den üblichen entsprechen, verstehen
- Die Theorie und Praxis im Zusammenhang mit der Verwendung von Prioritätsheaps und Prioritätswarteschlangen verstehen
- Die Funktionsweise von Hash-Tabellen als abstrakte Datentypen und Funktionen kennenlernen
- Die Graphentheorie sowie fortgeschrittene Graph-Algorithmen und Konzepte verstehen

Modul 3. Algorithmen und Komplexität

- Erlernen der wichtigsten Strategien für den Entwurf von Algorithmen sowie der verschiedenen Methoden und Maße für die Berechnung von Algorithmen
- Kenntnis der wichtigsten Sortieralgorithmen, die in der Softwareentwicklung verwendet werden
- Verstehen, wie verschiedene Algorithmen mit Bäumen, *Heaps* und Graphen arbeiten
- *Greedy-Algorithmen*, ihre Strategie und Beispiele für ihre Anwendung bei den wichtigsten bekannten Problemen verstehen
- Die Anwendung von *Greedy-Algorithmen* auf Graphen ebenfalls kennen
- Die wichtigsten Strategien der Suche nach minimalen Pfaden lernen, mit dem Ansatz der wesentlichen Probleme des Bereichs und Algorithmen für ihre Lösung
- Verstehen der *Backtracking*-Technik und ihrer wichtigsten Anwendungen sowie anderer alternativer Techniken

Modul 4. Fortgeschrittener Algorithmusentwurf

- Vertiefung in den fortgeschrittenen Entwurf von Algorithmen, Analyse von rekursiven Algorithmen und Divide-and-Conquer-Algorithmen sowie Durchführung von amortisierten Analysen
- Konzepte der dynamischen Programmierung und Algorithmen für NP-Probleme verstehen
- Die Funktionsweise der kombinatorischen Optimierung, sowie die verschiedenen Randomisierungsalgorithmen und parallelen Algorithmen verstehen
- Die Funktionsweise der verschiedenen lokalen und Kandidaten-Suchmethoden kennen und verstehen
- Erlernen der Mechanismen der formalen Verifikation von Programmen und iterativen Programmen, einschließlich der Logik erster Ordnung und des formalen Systems von Hoare
- Die Funktionsweise einiger der wichtigsten numerischen Methoden wie die Bisektionsmethode, die Newton-Raphson-Methode und die Sekantenmethode kennen lernen

Modul 5. Fortgeschrittene Programmierung

- Vertiefung der Kenntnisse in der Programmierung, insbesondere in Bezug auf die objektorientierte Programmierung, und der verschiedenen Arten von Beziehungen zwischen bestehenden Klassen
- Die verschiedenen Entwurfsmuster für objektorientierte Probleme kennenlernen
- Lernen der ereignisgesteuerte Programmierung und die Entwicklung von Benutzeroberflächen mit Qt kennen
- Grundlegende Kenntnisse über nebenläufige Programmierung, Prozesse und Threads erwerben
- Die Verwendung von Threads und Synchronisierung sowie die Lösung gängiger Probleme bei der gleichzeitigen Programmierung
- Die Bedeutung von Dokumentation und Tests bei der Softwareentwicklung verstehen

Modul 6. Theoretische Informatik

- Verstehen der wesentlichen theoretischen mathematischen Konzepte der Informatik, wie Aussagenlogik, Mengenlehre, nummerierbare und nicht nummerierbare Mengen
- Die Konzepte von formalen Sprachen und Grammatiken sowie von Turing-Maschinen in ihren verschiedenen Varianten verstehen
- Die verschiedenen Arten von unentscheidbaren und unlösbaren Problemen kennenlernen, einschließlich ihrer verschiedenen Varianten und Annäherungen
- Die Funktionsweise verschiedener randomisierungsbasierter Sprachen und anderer Arten von Klassen und Grammatiken verstehen
- Informationen über andere fortschrittliche Rechensysteme wie Membrane Computing, DNA Computing und Quantum Computing

Modul 7. Automatentheorie und formalen Sprachen

- Verständnis der Automatentheorie und der formalen Sprachen, Erlernen der Konzepte von Alphabeten, Zeichenketten und Sprachen sowie der Durchführung von formalen Demonstrationen
- Die verschiedenen Arten von endlichen Automaten kennenlernen, sowohl deterministische als auch nicht-deterministische
- Die grundlegenden und fortgeschrittenen Konzepte im Zusammenhang mit Sprachen und regulären Ausdrücken sowie die Anwendung des Pumping-Lemmas und die Schließung regulärer Sprachen erlernen
- Verstehen kontextunabhängiger Grammatiken sowie der Funktionsweise von Stapelautomaten
- Vertiefung der Normalformen, des Pump-Lemmas der kontextunabhängigen Grammatiken und der Eigenschaften kontextunabhängiger Sprachen

Modul 8. Sprachprozessoren

- Einführung in die Konzepte im Zusammenhang mit dem Kompilierungsprozess und den verschiedenen Arten der Analyse: lexikalisch, syntaktisch und semantisch
- Wissen, wie ein lexikalischer Analysator funktioniert, seine Implementierung und Fehlerbehebung
- Vertiefung der Kenntnisse der syntaktischen Analyse, sowohl top-down als auch bottom-up, aber mit besonderem Schwerpunkt auf den verschiedenen Arten von bottom-up syntaktischen Analysatoren
- Verstehen, wie semantische Parser funktionieren, die syntaxorientierte Tradition, die Symboltabelle und die verschiedenen Arten von Parsern
- Die verschiedenen Mechanismen der Codegenerierung kennenlernen, sowohl in Laufzeitumgebungen als auch für die Generierung von Zwischencode
- Die Grundlagen der Code-Optimierung, einschließlich der Neuordnung von Ausdrücken und der Optimierung von Schleifen

Modul 9. Computergrafik und Visualisierung

- Einführung in die grundlegenden Konzepte der Computergrafik und Computervisualisierung, wie z. B. die Farbtheorie und ihre Modelle sowie die Eigenschaften des Lichts
- Die Funktionsweise der Ausgabeprimitive und ihrer Algorithmen zu verstehen, sowohl für das Zeichnen von Linien als auch für das Zeichnen von Kreisen und Füllungen
- Vertiefung des Studiums der verschiedenen Transformationen, sowohl 2D als auch 3D, und ihrer Koordinatensysteme und Computervisualisierung
- Das Erstellen von Projektionen und Schnitten in 3D sowie das Eliminieren von verdeckten Flächen erlernen
- Die Theorie der Interpolation und der parametrischen Kurven sowie der Bézier-Kurven und B-Splines kennenlernen

Modul 10. Bio-inspiriertes Rechnen

- Einführung in das Konzept des Bio-Inspired Computing sowie Verständnis für die Funktionsweise verschiedener Arten von sozialen Anpassungsalgorithmen und genetischen Algorithmen
- Vertiefung des Studiums der verschiedenen Modelle des Evolutionary Computing, Kenntnis ihrer Strategien, Programmierung, Algorithmen und Modelle, die auf der Schätzung von Verteilungen basieren
- Die wichtigsten Strategien zur Erkundung und Ausnutzung des Raums für genetische Algorithmen verstehen
- Die Funktionsweise der evolutionären Programmierung bei Lernproblemen und Mehrzielproblemen verstehen
- Die grundlegenden Konzepte neuronaler Netze kennen und verstehen Sie die Funktionsweise realer Anwendungsfälle in so unterschiedlichen Bereichen wie medizinische Forschung, Wirtschaft und maschinelles Sehen erlernen

03

Kursleitung

Dieses akademische Programm verfügt über den spezialisiertesten Lehrkörper auf dem aktuellen Bildungsmarkt. Es handelt sich um Spezialisten, die von TECH ausgewählt wurden, um den gesamten Studiengang zu entwickeln. Auf diese Weise haben sie auf der Grundlage ihrer eigenen Erfahrung und der neuesten Erkenntnisse die aktuellsten Inhalte entworfen, die eine Qualitätsgarantie für ein so relevantes Thema bieten.

TECH bietet Ihnen den spezialisiertesten Lehrkörper in diesem Fachgebiet. Schreiben Sie sich jetzt ein und genießen Sie die Qualität, die Sie verdienen"

Internationaler Gastdirektor

Dr. Jeremy Gibbons gilt als internationale Eminenz für seine Beiträge im Bereich der Programmiermethodik und ihrer Anwendungen im Software Engineering. Seit mehr als zwei Jahrzehnten treibt dieser mit dem Fachbereich Informatik der Universität von Oxford verbundene Experte verschiedene Entwicklungsprojekte voran, deren greifbarste Ergebnisse von Informatikern in verschiedenen Teilen der Welt angewendet werden.

Seine Arbeit umfasst Bereiche wie generische Programmierung, formale Methoden, computergestützte Biologie, Bioinformatik und Algorithmenentwurf mit Haskell. Letzteres wurde in Zusammenarbeit mit seinem Mentor, Dr. Richard Bird, umfassend entwickelt.

In seiner Funktion als Direktor der Programmieralgebra-Forschungsgruppe hat Gibbons die Fortschritte bei den funktionalen Programmiersprachen und der Mustertheorie in der Programmierung vorangetrieben. Gleichzeitig wurden Anwendungen seiner Innovationen mit dem Gesundheitswesen in Verbindung gebracht, wie seine Zusammenarbeit mit CancerGrid und Datatype Generic Programming beweist. Diese und andere Initiativen wiederum spiegeln sein Interesse an der Lösung praktischer Probleme in der Krebsforschung und der klinischen Informatik wider.

Gibbons hat sich auch als Chefredakteur von wissenschaftlichen Veröffentlichungen in The Journal of Functional Programming und The Programming Journal: The Art, Science, and Engineering of Programming einen Namen gemacht. Im Rahmen dieser Aufgaben hat er sich intensiv um die Verbreitung von Wissen gekümmert. Darüber hinaus hatte er mehrere Lehrstühle inne, die mit renommierten Einrichtungen wie der Universität Oxford Brookes und der Universität von Auckland, Neuseeland, verbunden sind.

Darüber hinaus ist dieser Spezialist Mitglied der Arbeitsgruppe 2.1 über Algorithmische Sprachen und Berechnungen der International Federation for Information Processing (IFIP). Bei dieser Organisation ist er für die Wartung der Programmiersprachen ALGOL 60 und ALGOL 68 zuständig.

Dr. Gibbons Jeremy

- Direktor des Software-Engineering-Programms an der Universität von Oxford, UK
- Stellvertretender Leiter des Informatiklabors und der Fakultät für Informatik, Universität von Oxford
- Professor am Kellogg College, an der Universität Oxford Brookes und an der Universität von Auckland in Neuseeland
- Direktor der Forschungsgruppe Programmieralgebra
- Redakteur und Chef der Zeitschriften The Art, Science, and Engineering of Programming und Journal of Functional Programming
- Promotion in Computerwissenschaften an der Universität von Oxford
- Masterstudiengang in Informatik an der Universität von Edinburgh
- Internationale Föderation für Informationsverarbeitung (IFIP) Arbeitsgruppe 2.1 über Algorithmische Sprachen und Berechnungen (WG2.1)

Dank TECH werden Sie mit den besten Fachleuten der Welt lernen können"

04

Kompetenzen

Nach Bestehen der Prüfungen des Programms in Informatik und Sprachen wird die Fachkraft die notwendigen Kompetenzen erworben haben, um die grundlegenden Prinzipien der Informatik zu kennen und mit Programmiersprachen und Daten arbeiten zu können.

Erwerben Sie die Fähigkeit, neue Computerentwicklungen durchzuführen, die auf dem Verständnis und der Beherrschung der verschiedenen Sprachen und Algorithmen und ihrer praktischen Anwendung beruhen“

Allgemeine Kompetenz

- Korrekte Ausführung von Aufgaben im Zusammenhang mit Computern und Computersprache

Verbessern Sie Ihre Fähigkeiten, um an verschiedenen technologischen Projekten teilzunehmen“

Spezifische Kompetenzen

- Algorithmen zur Entwicklung von Computerprogrammen entwerfen und die Programmiersprache anwenden
- Computerdatenstruktur verstehen und verwenden
- Algorithmen verwenden, die zur Lösung von Computerproblemen benötigt werden
- Gründliches Verständnis von fortgeschrittenem Algorithmusdesign und Suchmethoden
- Durchführung von Aufgaben der Computerprogrammierung
- Verstehen und Anwenden der theoretischen Grundlagen der Informatik, z. B. der Mathematik
- Die Theorie der Computerautomaten kennen und die Computersprache anwenden können
- Verstehen der theoretischen Grundlagen von Programmiersprachen und der damit verbundenen lexikalischen, syntaktischen und semantischen Verarbeitungstechniken
- Die grundlegenden Konzepte der Mathematik und der rechnerischen Komplexität verstehen, um sie bei der Lösung von Computerproblemen anzuwenden
- Die grundlegenden Prinzipien der Informatik kennen und anwenden, um neue Computerentwicklungen durchzuführen

05

Struktur und Inhalt

Die Struktur der Inhalte ist so angelegt, dass das Wissen schrittweise aufgenommen wird, um einen Wachstumspfad zu erreichen, der Sie zu Spitzenleistungen in Ihrem Beruf führen wird.

Alle Interessensgebiete, die Sie beherrschen müssen, um sicher und erfolgreich in den Bereichen Informatik und Programmiersprachen arbeiten zu können, sind in einem hochwertigen Lehrplan zusammengefasst“

Modul 1. Grundlagen der Programmierung

1.1. Einführung in die Programmierung
- 1.1.1. Grundlegende Struktur eines Computers
- 1.1.2. Software
- 1.1.3. Programmiersprachen
- 1.1.4. Lebenszyklus einer Softwareanwendung

1.2. Algorithmusentwurf
- 1.2.1. Lösung von Problemen
- 1.2.2. Deskriptive Techniken
- 1.2.3. Elemente und Struktur eines Algorithmus

1.3. Elemente eines Programms
- 1.3.1. Herkunft und Merkmale der Sprache C++
- 1.3.2. Die Entwicklungsumgebung
- 1.3.3. Konzept des Programms
- 1.3.4. Arten von grundlegender Daten
- 1.3.5. Betreiber
- 1.3.6. Ausdrücke
- 1.3.7. Sätze
- 1.3.8. Dateneingabe und -ausgabe

1. 4. Kontrollsätze
- 1.4.1. Sätze
- 1.4.2. Verzweigungen
- 1.4.3. Schleifen

1.5. Abstraktion und Modularität: Funktionen
- 1.5.1. Modularer Aufbau
- 1.5.2. Konzept der Funktion und des Nutzens
- 1.5.3. Definition einer Funktion
- 1.5.4. Ausführungsablauf beim Aufruf einer Funktion
- 1.5.5. Prototyp einer Funktion
- 1.5.6. Rückgabe der Ergebnisse
- 1.5.7. Aufrufen einer Funktion: Parameter
- 1.5.8. Übergabe von Parametern per Referenz und per Wert
- 1.5.9. Kennung des Geltungsbereichs

1.6. Statische Datenstrukturen
- 1.6.1. *Arrays*
- 1.6.2. Matrizen. Polyeder
- 1.6.3. Suchen und Sortieren
- 1.6.4. Zeichenketten. E/A-Funktionen für Zeichenketten
- 1.6.5. Strukturen. Verbindungen
- 1.6.6. Neue Datentypen

1.7. Dynamische Datenstrukturen: Zeiger
- 1.7.1. Konzept. Definition von Zeiger
- 1.7.2. Operatoren und Operationen mit Zeigern
- 1.7.3. *Arrays* von Zeigern
- 1.7.4. Zeiger und *Arrays*
- 1.7.5. Zeiger auf Zeichenketten
- 1.7.6. Zeiger auf Strukturen
- 1.7.7. Multiple Indirektion
- 1.7.8. Zeiger auf Funktionen
- 1.7.9. Übergabe von Funktionen, Strukturen und *Arrays* als Funktionsparameter

1.8. Dateien
- 1.8.1. Grundlegende Konzepte
- 1.8.2. Dateioperationen
- 1.8.3. Datentypen
- 1.8.4. Organisation von Dateien
- 1.8.5. Einführung in C++ Dateien
- 1.8.6. Handhabung von Dateien

1.9. Rekursion
- 1.9.1. Definition von Rekursion
- 1.9.2. Arten der Rekursion
- 1.9.3. Vorteile und Nachteile
- 1.9.4. Überlegungen
- 1.9.5. Rekursiv-iterative Umwandlung
- 1.9.6. Der Rekursionsstapel

1.10. Prüfung und Dokumentation
- 1.10.1. Programm-Tests
- 1.10.2. White Box-Tests
- 1.10.3. Black Box-Tests
- 1.10.4. Test-Tools
- 1.10.5. Programm-Dokumentation

Modul 2. Datenstruktur

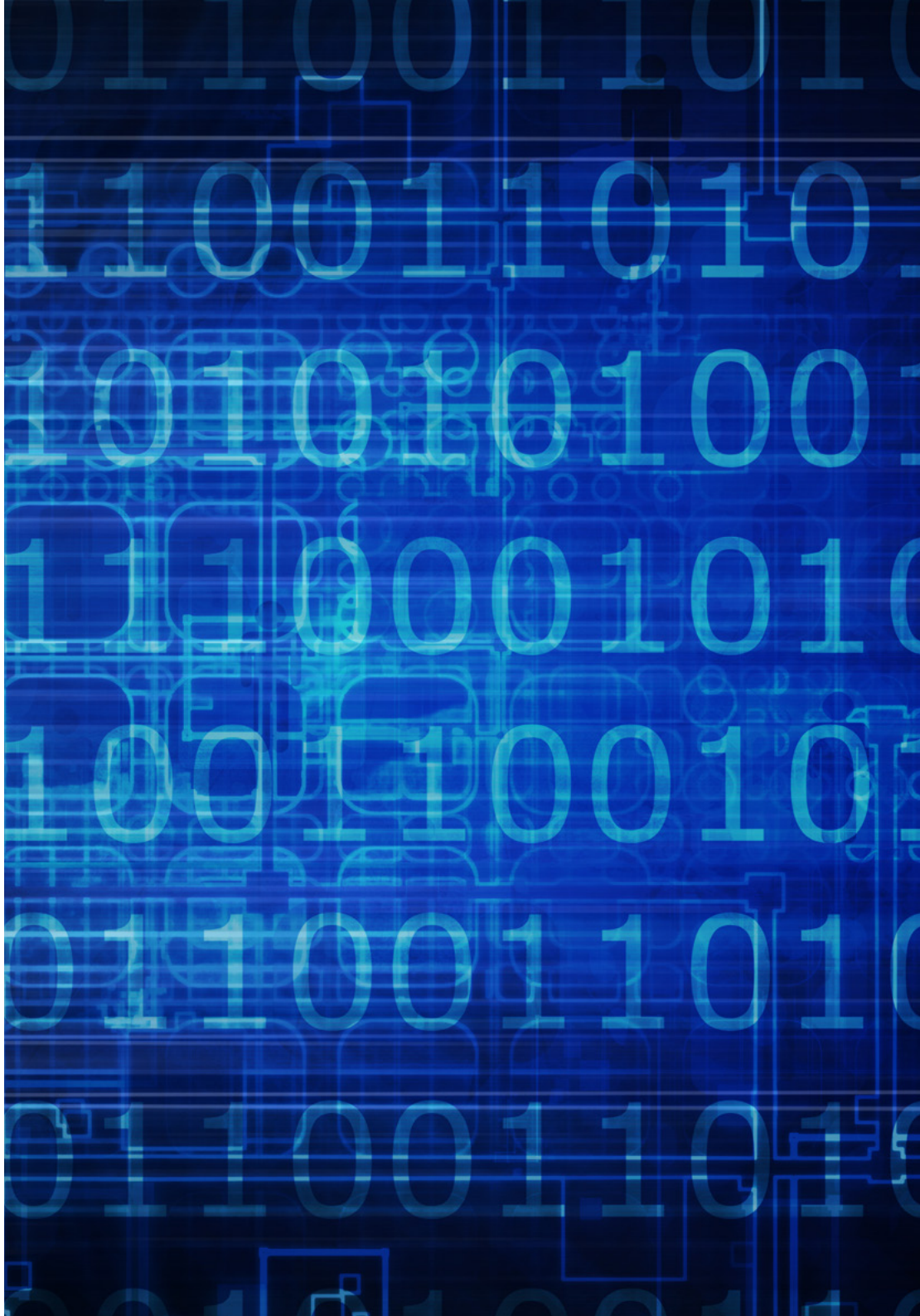
2.1. Einführung in die Programmierung in C++
- 2.1.1. Klassen, Konstruktoren, Methoden und Attribute
- 2.1.2. Variablen
- 2.1.3. Bedingte Ausdrücke und Schleifen
- 2.1.4. Objekte

2.2. Abstrakte Datentypen (ADT)
- 2.2.1. Datentypen
- 2.2.2. Grundlegende Strukturen und ADTs
- 2.2.3. Vektoren und Arrays

2.3. Lineare Datenstrukturen
- 2.3.1. ADT-Liste. Definition
- 2.3.2. Verknüpfte und doppelt verknüpfte Listen
- 2.3.3. Geordnete Listen
- 2.3.4. Listen in C++
- 2.3.5. ADT-Stack
- 2.3.6. ADT-Queues
- 2.3.7. Stack und Queue in C++

2.4. Hierarchische Datenstrukturen
- 2.4.1. ADT-Baum
- 2.4.2. Pfade
- 2.4.3. n-äre Bäume
- 2.4.4. Binäre Bäume
- 2.4.5. Binäre Suchbäume

2.5. Hierarchische Datenstrukturen: Komplexe Bäume
  2.5.1. Perfekt ausbalancierte oder minimal hohe Bäume
  2.5.2. Bäume mit mehreren Pfaden
  2.5.3. Bibliografische Referenzen
2.6. Heaps und Prioritätswarteschlange
  2.6.1. ADT-Heaps
  2.6.2. ADT Vorrangige Queue
2.7. Hash-Tabellen
  2.7.1. ADT in *Hash*-Tabellen
  2.7.2. *Hash*-Funktionen
  2.7.3. *Hash*-Funktion in *Hash*-Tabellen
  2.7.4. Redispersion
  2.7.5. Offene *Hash*-Tabellen
2.8. Graph
  2.8.1. ADT-Graph
  2.8.2. Graph Typen
  2.8.3. Grafische Darstellung und Grundoperationen
  2.8.4. Netzwerk Design
2.9. Graph-Algorithmen und fortgeschrittene Graph-Konzepte
  2.9.1. Graph Probleme
  2.9.2. Pfad-Algorithmen
  2.9.3. Such- oder Pfad-Algorithmen
  2.9.4. Andere Algorithmen
2.10. Andere Datenstrukturen
  2.10.1. Sets
  2.10.2. Parallele Arrays
  2.10.3. Symboltabellen
  2.10.4. *Tries*



Modul 3. Algorithmen und Komplexität

3.1. Einführung in Algorithmus-Design-Strategien
- 3.1.1. Rekursion
- 3.1.2. Aufteilen und erobern
- 3.1.3. Andere Strategien

3.2. Effizienz und Analyse von Algorithmen
- 3.2.1. Maßnahmen zur Steigerung der Effizienz
- 3.2.2. Messung der Eingabegröße
- 3.2.3. Messung der Ausführungszeit
- 3.2.4. Schlimmster, bester und durchschnittlicher Fall
- 3.2.5. Asymptotische Notation
- 3.2.6. Kriterien für die mathematische Analyse von nicht-rekursiven Algorithmen
- 3.2.7. Mathematische Analyse von rekursiven Algorithmen
- 3.2.8. Empirische Analyse von Algorithmen

3.3. Sortieralgorithmen
- 3.3.1. Konzept der Sortierung
- 3.3.2. Blase sortieren
- 3.3.3. Sortieren nach Auswahl
- 3.3.4. Reihenfolge der Insertion
- 3.3.5. Sortieren durch zusammenführen (Merge Sort)
- 3.3.6. Schnelle Sortierung (Quick Sort)

3.4. Algorithmen mit Bäumen
- 3.4.1. Konzept des Baumes
- 3.4.2. Binäre Bäume
- 3.4.3. Baumpfade
- 3.4.4. Ausdrücke darstellen
- 3.4.5. Geordnete binäre Bäume
- 3.4.6. Ausgeglichene binäre Bäume

3.5. Algorithmen mit *Heaps*
- 3.5.1. *Heaps*
- 3.5.2. Der HeapSort Algorithmus
- 3.5.3. Prioritätswarteschlangen

3.6. Graph Algorithmen
- 3.6.1. Vertretung
- 3.6.2. Lauf in Breite
- 3.6.3. Lauf in Tiefe
- 3.6.4. Topologische Anordnung

3.7. *Greedy*-Algorithmen
- 3.7.1. Die *Greedy*-Strategie
- 3.7.2. Elemente der *Greedy*-Strategie
- 3.7.3. Währungsumtausch
- 3.7.4. Das Problem des Reisenden
- 3.7.5. Problem mit dem Rucksack

3.8. Minimale Pfadsuche
- 3.8.1. Das Problem des minimalen Pfades
- 3.8.2. Negative Bögen und Zyklen
- 3.8.3. Dijkstras Algorithmus

3.9. Greedy Algorithmen auf Graphs
- 3.9.1. Der minimal aufspannende Baum
- 3.9.2. Prims Algorithmus
- 3.9.3. Kruskals Algorithmus
- 3.9.4. Komplexitätsanalyse

3.10. *Backtracking*
- 3.10.1. Das *Backtracking*
- 3.10.2. Alternative Techniken

Modul 4. Fortgeschrittener Algorithmusentwurf

4.1. Analyse von rekursiven und Teilen-und-Erobern-Algorithmen
- 4.1.1. Aufstellen und Lösen von homogenen und nicht-homogenen Rekursionsgleichungen
- 4.1.2. Überblick über die Strategie des Teilens und Eroberns

4.2. Amortisierte Analyse
- 4.2.1. Aggregierte Analyse
- 4.2.2. Die Buchhaltungsmethode
- 4.2.3. Die potenzielle Methode

4.3. Dynamische Programmierung und Algorithmen für NP-Probleme
- 4.3.1. Merkmale der dynamischen Programmierung
- 4.3.2. Rückverfolgung: backtracking
- 4.3.3. Verzweigung und Beschneidung

4.4. Kombinatorische Optimierung
- 4.4.1. Problemdarstellung
- 4.4.2. 1D-Optimierung

4.5. Randomisierungsalgorithmen
- 4.5.1. Beispiele für Randomisierungsalgorithmen
- 4.5.2. Das Buffonsche Theorem
- 4.5.3. Monte-Carlo-Algorithmus
- 4.5.4. Las Vegas Algorithmus

4.6. Lokale Suche und Kandidatensuche
- 4.6.1. *Garcient Ascent*
- 4.6.2. *Hill Climbing*
- 4.6.3. *Simulated Annealing*
- 4.6.4. *Tabu Search*
- 4.6.5. Suche mit Kandidaten

4.7. Formale Überprüfung von Programmen
- 4.7.1. Spezifikation von funktionalen Abstraktionen
- 4.7.2. Die Sprache der Logik erster Ordnung
- 4.7.3. Hoare's formales System

4.8. Verifizierung von iterativen Programmen
- 4.8.1. Regeln des formalen Hoare-Systems
- 4.8.2. Konzept der invarianten Iterationen

4.9. Numerische Methoden
- 4.9.1. Die Methode der Halbierung
- 4.9.2. Die Newton-Raphson-Methode
- 4.9.3. Die Sekantenmethode

4.10. Parallele Algorithmen
- 4.10.1. Parallele binäre Operationen
- 4.10.2. Parallele Operationen mit Diagrammen
- 4.10.3. Parallelität in Teilen und Erobern
- 4.10.4. Parallelität in der dynamischen Programmierung

Modul 5. Fortgeschrittene Programmierung

5.1. Einführung in die objektorientierte Programmierung
- 5.1.1. Einführung in die objektorientierte Programmierung
- 5.1.2. Klassen-Design
- 5.1.3. Einführung in UML für die Modellierung von Problemen

5.2. Beziehungen zwischen Klassen
- 5.2.1. Abstraktion und Vererbung
- 5.2.2. Fortgeschrittene Konzepte der Vererbung
- 5.2.3. Polymorphismen
- 5.2.4. Zusammensetzung und Aggregation

5.3. Einführung in Design Patterns für objektorientierte Probleme
- 5.3.1. Was sind Entwurfsmuster
- 5.3.2. *Factory*-Muster
- 5.3.3. *Singleton*-Muster
- 5.3.4. *Observer*-Muster
- 5.3.5. *Composite*-Muster

5.4. Ausnahmen
- 5.4.1. Was sind Ausnahmen?
- 5.4.2. Abfangen und Behandlung von Ausnahmen
- 5.4.3. Werfen von Ausnahmen
- 5.4.4. Erstellung von Ausnahmen

5.5. Benutzeroberflächen
- 5.5.1. Einführung in Qt
- 5.5.2. Positionierung

5.5.3. Was sind Ereignisse?
5.5.4. Ereignisse: Definition und Erfassung
5.5.5. Entwicklung von Benutzeroberflächen
5.6. Einführung in die gleichzeitige Programmierung
5.6.1. Einführung in die gleichzeitige Programmierung
5.6.2. Der Prozess und das Thread-Konzept
5.6.3. Interaktion zwischen Prozessen oder Threads
5.6.4. Threads in C++
5.6.5. Vor- und Nachteile der gleichzeitigen Programmierung
5.7. Thread-Verwaltung und Synchronisierung
5.7.1. Lebenszyklus eines Threads
5.7.2. Die Klasse *Thread*
5.7.3. Planung des Threads
5.7.4. Gruppen von Threads
5.7.5. Daemon Threads
5.7.6. Synchronisierung
5.7.7. Verriegelungsmechanismen
5.7.8. Kommunikationsmechanismen
5.7.9. Monitore
5.8. Häufige Probleme bei der gleichzeitigen Programmierung
5.8.1. Das Erzeuger-Verbraucher-Problem
5.8.2. Das Problem von Lesern und Schriftstellern
5.8.3. Das Problem mit dem Philosophenproblem
5.9. Dokumentation und Prüfung von Software
5.9.1. Warum ist es wichtig, Software zu dokumentieren?
5.9.2. Design-Dokumentation
5.9.3. Verwendung von Tools zur Dokumentation
5.10. Software-Tests
5.10.1. Einführung in die Softwareprüfung
5.10.2. Arten von Tests
5.10.3. Einheitstest
5.10.4. Integrationstests
5.10.5. Validierungstest
5.10.6. Systemprüfung

Modul 6. Theoretische Informatik

6.1. Verwendete mathematische Konzepte
6.1.1. Einführung in die Aussagenlogik
6.1.2. Theorie der Beziehungen
6.1.3. Abzählbare und nicht abzählbare Mengen
6.2. Formale Sprachen und Grammatiken und Einführung in Turingmaschinen
6.2.1. Formale Sprachen und Grammatiken
6.2.2. Problem der Entscheidung
6.2.3. Die Turingmaschine
6.3. Erweiterungen für Turing-Maschinen, eingeschränkte Turing-Maschinen und Computer
6.3.1. Programmiertechniken für Turingmaschinen
6.3.2. Erweiterungen für Turingmaschinen
6.3.3. Eingeschränkte Turingmaschinen
6.3.4. Turingmaschinen und Computer
6.4. Unsagbarkeit
6.4.1. Nicht rekursiv aufzählbare Sprache
6.4.2. Ein rekursiv aufzählbares unsagbares Problem
6.5. Andere unsagbares Probleme
6.5.1. Unsagbare Probleme für Turingmaschinen
6.5.2. Postkorrespondenz-Problem (PCP)
6.6. Unlösbare Probleme
6.6.1. Die Klassen P und NP
6.6.2. Ein NP-komplettes Problem
6.6.3. Problem der eingeschränkten Erfüllbarkeit
6.6.4. Andere NP-komplette Probleme
6.7. Co-NP und PS Probleme
6.7.1. Komplementär zu NP-Sprachen
6.7.2. Im Polynomraum lösbare Probleme
6.7.3. Vollständige PS-Probleme

6.8. Klassen von randomisierungsbasierten Sprachen
- 6.8.1. MT-Modell mit Randomisierung
- 6.8.2. Die Klassen RP und ZPP
- 6.8.3. Primzahl-Test
- 6.8.4. Komplexität des Primzahltests

6.9. Andere Klassen und Grammatiken
- 6.9.1. Probabilistische endliche Automaten
- 6.9.2. Zelluläre Automaten
- 6.9.3. McCullogh- und Pitts-Zellen
- 6.9.4. Lindenmayer-Grammatiken

6.10. Fortgeschrittene Rechensysteme
- 6.10.1. Berechnung der Membranen: P-Systeme
- 6.10.2. DNA-Computing
- 6.10.3. Quantencomputing

Modul 7. Automatentheorie und formalen Sprachen

7.1. Einführung in die Automatentheorie
- 7.1.1. Warum Automaten-Theorie studieren?
- 7.1.2. Einführung in formale Beweise
- 7.1.3. Andere Formen des Nachweises
- 7.1.4. Mathematische Induktion
- 7.1.5. Alphabete, Zeichenketten und Sprachen

7.2. Deterministische endliche Automaten
- 7.2.1. Einführung in endliche Automaten
- 7.2.2. Deterministische endliche Automaten

7.3. Nicht-deterministische endliche Automaten
- 7.3.1. Nicht-deterministische endliche Automaten
- 7.3.2. Äquivalenz zwischen AFD und AFN
- 7.3.3. Endliche Automaten mit Übergängen

7.4. Sprachen und reguläre Begriffe (I)
- 7.4.1. Sprachen und reguläre Begriffe
- 7.4.2. Endliche Automaten und reguläre Begriffe

7.5. Sprachen und reguläre Begriffe (II)
- 7.5.1. Umwandlung von regulären Begriffen in Automaten
- 7.5.2. Anwendungen von regulären Begriffen
- 7.5.3. Algebra der regulären Begriffe

7.6. Pumping-Lemma und Abschluss von regulären Sprachen
- 7.6.1. Pumping-Lemma
- 7.6.2. Abschlusseigenschaften von regulären Sprachen

7.7. Äquivalenz und Minimierung von Automaten
- 7.7.1. FA Äquivalenz
- 7.7.2. FA Minimierung

7.8. Kontextunabhängige Grammatiken (KUG)
- 7.8.1. Kontextunabhängige Grammatiken
- 7.8.2. Ableitungsbäume
- 7.8.3. Anwendungen von KUGs
- 7.8.4. Mehrdeutigkeit in Grammatiken und Sprachen

7.9. Stapelautomaten und GICs
- 7.9.1. Definition von Stapelautomaten
- 7.9.2. Von einem Stapelautomaten akzeptierte Sprachen
- 7.9.3. Äquivalez zwischen Stapelautomaten und Kontextfreie Grammatik
- 7.9.4. Deterministischer Stapelautomaten

7.10. Normalformen, Pump-Lemma von KUGs und Eigenschaften von LICs
- 7.10.1. Normale Formen von KUGs
- 7.10.2. Pumping-Lemma
- 7.10.3. Abschlusseigenschaften von regulären Sprachen
- 7.10.4. Entscheidungseigenschaften von LICs

Modul 8. Sprachprozessoren

8.1. Einführung in den Kompilierungsprozess
- 8.1.1. Zusammenstellung und Interpretation
- 8.1.2. Compiler-Ausführungsumgebung
- 8.1.3. Analyse-Prozess
- 8.1.4. Prozess der Synthese

8.2. Lexikalischer Analysator
- 8.2.1. Was ist ein lexikalischer Analysator?
- 8.2.2. Implementierung des lexikalischen Analysators
- 8.2.3. Semantische Aktionen
- 8.2.4. Fehlerbehebung
- 8.2.5. Fragen der Implementierung

8.3. Syntaktische Analyse
- 8.3.1. Was ist ein Parser?
- 8.3.2. Vorläufige Konzepte
- 8.3.3. Top-down-Parser
- 8.3.4. Bottom-up-Analysatoren

8.4. Top-down-Parsing und Bottom-up-Parsing
- 8.4.1. LL(1) Analysator
- 8.4.2. LR(0) Analysator
- 8.4.3. Beispiel für einen Parser

8.5. Erweitertes Bottom-up-Parsing
- 8.5.1. SLR-Parser
- 8.5.2. LR-Parser(1)
- 8.5.3. LR-Parser (k)
- 8.5.4. LALR-Parser

8.6. Semantische Analyse (I)
- 8.6.1. Syntaxgesteuerte Übersetzung
- 8.6.2. Symboltabelle

8.7. Semantische Analyse (II)
- 8.7.1. Typenprüfung
- 8.7.2. Das Subsystem Typ
- 8.7.3. Typ-Äquivalenz und Konvertierungen

8.8. Codegenerierung und Ausführungsumgebung
- 8.8.1. Design-Aspekte
- 8.8.2. Ausführungsumgebung
- 8.8.3. Speicherorganisation
- 8.8.4. Speicherzuweisung

8.9. Zwischencodegenerierung
- 8.9.1. Synthesegesteuerte Übersetzung
- 8.9.2. Zwischendarstellungen
- 8.9.3. Beispiele für Übersetzungen

8.10. Code-Optimierung
- 8.10.1. Register Zuweisung
- 8.10.2. Eliminierung toter Zuweisungen
- 8.10.3. Ausführung zur Kompilierzeit
- 8.10.4. Neuordnung von Ausdrücken
- 8.10.5. Schleifen-Optimierung

Modul 9. Computergrafik und Visualisierung

9.1. Farbtheorie
- 9.1.1. Eigenschaften von Licht
- 9.1.2. Farbige Modelle
- 9.1.3. Der CIE-Standard
- 9.1.4. *Profiling*

9.2. Ausgabe-Primitive
- 9.2.1. Der Videotreiber
- 9.2.2. Algorithmen zum Zeichnen von Linien
- 9.2.3. Algorithmen zum Zeichnen von Kreisen
- 9.2.4. Algorithmen zum Füllen

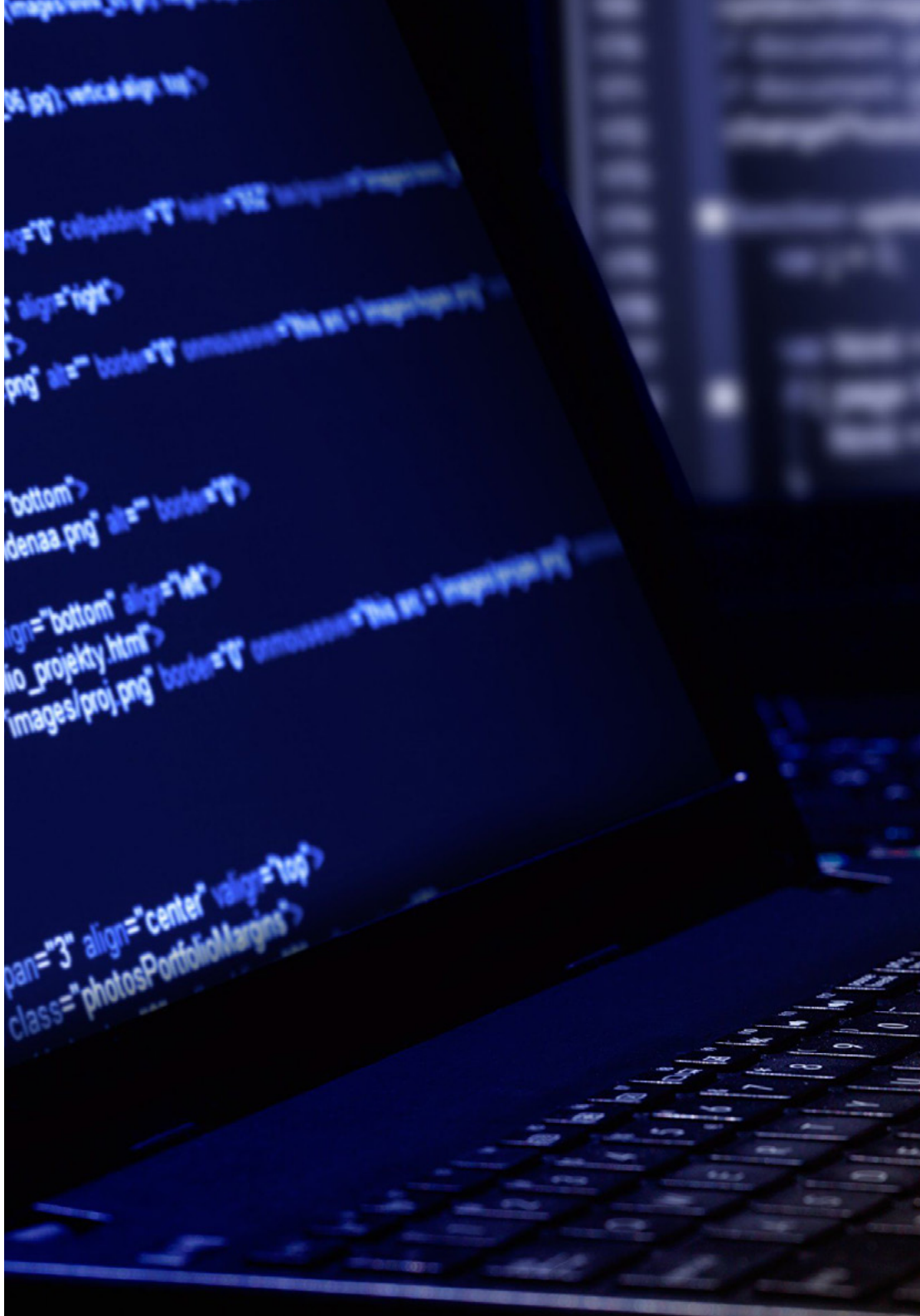
9.3. 2D-Transformationen und 2D-Koordinatensysteme und 2D-Clipping
- 9.3.1. Geometrische Grundtransformationen
- 9.3.2. Homogene Koordinaten
- 9.3.3. Inverse Transformation
- 9.3.4. Komposition von Transformationen
- 9.3.5. Andere Transformationen
- 9.3.6. Koordinate ändern
- 9.3.7. 2D-Koordinatensysteme
- 9.3.8. Koordinatenverschiebung
- 9.3.9. Normalisierung
- 9.3.10. Algorithmen zum Trimmen

9.4. 3D-Transformationen
- 9.4.1. Übertragung
- 9.4.2. Rotation
- 9.4.3. Skalierung
- 9.4.4. Reflexion
- 9.4.5. Scheren

9.5. Anzeige und Änderung von 3D-Koordinaten
- 9.5.1. 3D-Koordinatensysteme
- 9.5.2. Visualisierung
- 9.5.3. Koordinatenverschiebung
- 9.5.4. Projektion und Normalisierung

9.6. Projektion und 3D-Clipping
- 9.6.1. Orthogonale Projektion
- 9.6.2. Schräge Parallelprojektion
- 9.6.3. Perspektivische Projektion
- 9.6.4. 3D-Beschneidungsalgorithmen

9.7. Entfernen von verdeckten Flächen
- 9.7.1. *Back face removal*
- 9.7.2. Z-buffer
- 9.7.3. Algorithmus des Malers
- 9.7.4. Warnock-Algorithmus
- 9.7.5. Erkennung verdeckter Linien



9.8. Interpolation und parametrische Kurven
- 9.8.1. Interpolation und polynomielle Approximation
- 9.8.2. Parametrische Darstellung
- 9.8.3. Lagrange-Polynom
- 9.8.4. Natürliche kubische *Splines*
- 9.8.5. Basis-Funktionen
- 9.8.6. Matrix-Darstellung

9.9. Bézier-Kurven
- 9.9.1. Algebraische Konstruktion
- 9.9.2. Matrix-Formular
- 9.9.3. Zusammensetzung
- 9.9.4. Geometrische Konstruktion
- 9.9.5. Algorithmus zum Zeichnen

9.10. *B-Splines*
- 9.10.1. Das Problem der lokalen Kontrolle
- 9.10.2. Gleichmäßige kubische B-Splines
- 9.10.3. Basisfunktionen und Kontrollpunkte
- 9.10.4. Ableitung zum Ursprung und Multiplizität
- 9.10.5. Matrix-Darstellung
- 9.10.6. Uneinheitliche*B-Splines*

Modul 10. Bio-inspiriertes Rechnen

10.1. Einführung in das Bio-Inspired Computing
- 10.1.1. Einführung in das Bio-Inspired Computing

10.2. Algorithmen zur sozialen Anpassung
- 10.2.1. Bio-inspiriertes Rechnen auf der Basis von Ameisenkolonien
- 10.2.2. Varianten von Ameisenkolonie-Algorithmen
- 10.2.3. Cloud-basiertes Computing auf Partikelebene

10.3. Genetische Algorithmen
- 10.3.1. Allgemeine Struktur
- 10.3.2. Implementierungen der wichtigsten Operatoren

10.4. Explorations-Ausbeutungsraum-Strategien für genetische Algorithmen
- 10.4.1. CHC-Algorithmus
- 10.4.2. Multimodale Probleme

10.5. Evolutionäre Berechnungsmodelle (I)
- 10.5.1. Evolutionäre Strategien
- 10.5.2. Evolutionäre Programmierung
- 10.5.3. Algorithmen auf der Grundlage der differentiellen Evolution

10.6. Evolutionäre Berechnungsmodelle (II)
- 10.6.1. Evolutionäre Modelle auf der Grundlage der Schätzung von Verteilungen (EDA)
- 10.6.2. Genetische Programmierung

10.7. Evolutionäre Programmierung angewandt auf Lernprobleme
- 10.7.1. Regelbasiertes Lernen
- 10.7.2. Evolutionäre Methoden bei Instanzauswahlproblemen

10.8. Multi-Objektive Probleme
- 10.8.1. Konzept der Dominanz
- 10.8.2. Anwendung evolutionärer Algorithmen auf multikriterielle Probleme

10.9. Neuronale Netze (I)
- 10.9.1. Einführung in neuronale Netzwerke
- 10.9.2. Praktisches Beispiel mit neuronalen Netzwerken

10.10. Neuronale Netze
- 10.10.1. Anwendungsbeispiele für neuronale Netze in der medizinischen Forschung
- 10.10.2. Anwendungsbeispiele für neuronale Netze in der Wirtschaft
- 10.10.3. Anwendungsfälle für neuronale Netze in der industriellen Bildverarbeitung

06

Methodik

Dieses Fortbildungsprogramm bietet eine andere Art des Lernens. Unsere Methodik wird durch eine zyklische Lernmethode entwickelt: ***das Relearning***.

Dieses Lehrsystem wird z. B. an den renommiertesten medizinischen Fakultäten der Welt angewandt und wird von wichtigen Publikationen wie dem ***New England Journal of Medicine*** als eines der effektivsten angesehen.

Entdecken Sie Relearning, ein System, das das herkömmliche lineare Lernen aufgibt und Sie durch zyklische Lehrsysteme führt: eine Art des Lernens, die sich als äußerst effektiv erwiesen hat, insbesondere in Fächern, die Auswendiglernen erfordern"

Fallstudie zur Kontextualisierung aller Inhalte

Unser Programm bietet eine revolutionäre Methode zur Entwicklung von Fähigkeiten und Kenntnissen. Unser Ziel ist es, Kompetenzen in einem sich wandelnden, wettbewerbsorientierten und sehr anspruchsvollen Umfeld zu stärken.

Mit TECH werden Sie eine Art des Lernens erleben, die die Grundlagen der traditionellen Universitäten in der ganzen Welt verschiebt"

Sie werden Zugang zu einem Lernsystem haben, das auf Wiederholung basiert, mit natürlichem und progressivem Unterricht während des gesamten Lehrplans.

Die Studenten lernen durch gemeinschaftliche Aktivitäten und reale Fälle die Lösung komplexer Situationen in realen Geschäftsumgebungen.

Eine innovative und andersartige Lernmethode

Dieses TECH-Programm ist ein von Grund auf neu entwickeltes, intensives Lehrprogramm, das die anspruchsvollsten Herausforderungen und Entscheidungen in diesem Bereich sowohl auf nationaler als auch auf internationaler Ebene vorsieht. Dank dieser Methodik wird das persönliche und berufliche Wachstum gefördert und ein entscheidender Schritt in Richtung Erfolg gemacht. Die Fallmethode, die Technik, die diesem Inhalt zugrunde liegt, gewährleistet, dass die aktuellste wirtschaftliche, soziale und berufliche Realität berücksichtigt wird.

> *Unser Programm bereitet Sie darauf vor, sich neuen Herausforderungen in einem unsicheren Umfeld zu stellen und in Ihrer Karriere erfolgreich zu sein"*

Die Fallmethode ist das am weitesten verbreitete Lernsystem an den besten Informatikschulen der Welt, seit es sie gibt. Die Fallmethode wurde 1912 entwickelt, damit die Jurastudenten das Recht nicht nur anhand theoretischer Inhalte erlernen, sondern ihnen reale, komplexe Situationen vorlegen, damit sie fundierte Entscheidungen treffen und Werturteile darüber fällen können, wie diese zu lösen sind. Sie wurde 1924 als Standardlehrmethode in Harvard eingeführt.

Was sollte eine Fachkraft in einer bestimmten Situation tun? Mit dieser Frage konfrontieren wir Sie in der Fallmethode, einer handlungsorientierten Lernmethode. Während des gesamten Kurses werden die Studierenden mit mehreren realen Fällen konfrontiert. Sie müssen Ihr gesamtes Wissen integrieren, recherchieren, argumentieren und Ihre Ideen und Entscheidungen verteidigen.

Relearning Methodik

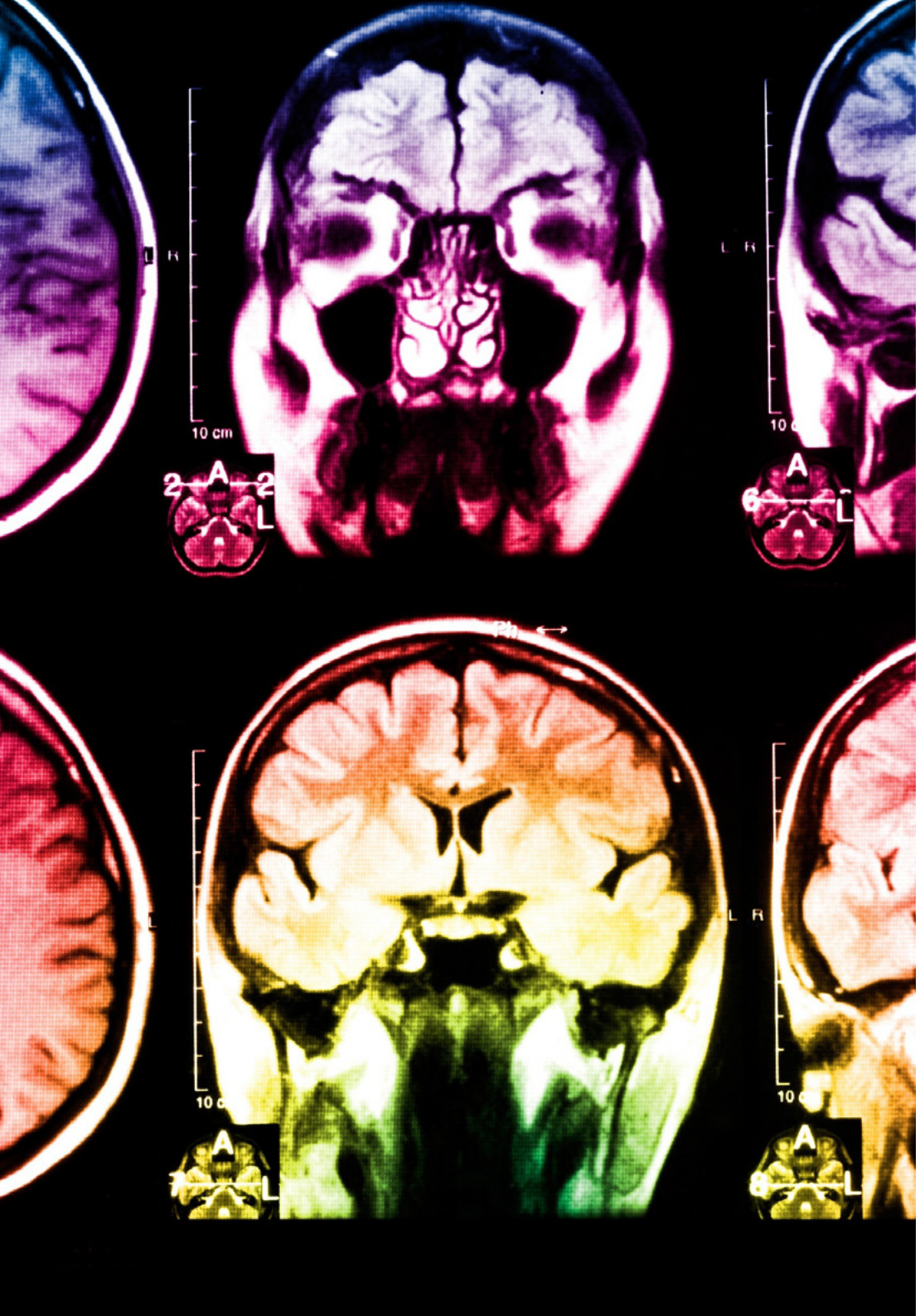
TECH kombiniert die Methodik der Fallstudien effektiv mit einem 100%igen Online-Lernsystem, das auf Wiederholung basiert und in jeder Lektion verschiedene didaktische Elemente kombiniert.

Wir ergänzen die Fallstudie mit der besten 100%igen Online-Lehrmethode: Relearning.

Im Jahr 2019 erzielten wir die besten Lernergebnisse aller spanischsprachigen Online-Universitäten der Welt.

Bei TECH lernen Sie mit einer hochmodernen Methodik, die darauf ausgerichtet ist, die Führungskräfte der Zukunft auszubilden. Diese Methode, die an der Spitze der weltweiten Pädagogik steht, wird Relearning genannt.

Unsere Universität ist die einzige in der spanischsprachigen Welt, die für die Anwendung dieser erfolgreichen Methode zugelassen ist. Im Jahr 2019 ist es uns gelungen, die Gesamtzufriedenheit unserer Studenten (Qualität der Lehre, Qualität der Materialien, Kursstruktur, Ziele...) in Bezug auf die Indikatoren der besten Online-Universität in Spanisch zu verbessern.



In unserem Programm ist das Lernen kein linearer Prozess, sondern erfolgt in einer Spirale (lernen, verlernen, vergessen und neu lernen). Daher wird jedes dieser Elemente konzentrisch kombiniert. Mit dieser Methode wurden mehr als 650.000 Hochschulabsolventen mit beispiellosem Erfolg in so unterschiedlichen Bereichen wie Biochemie, Genetik, Chirurgie, internationales Recht, Managementfähigkeiten, Sportwissenschaft, Philosophie, Recht, Ingenieurwesen, Journalismus, Geschichte, Finanzmärkte und -Instrumente ausgebildet. Dies alles in einem sehr anspruchsvollen Umfeld mit einer Studentenschaft mit hohem sozioökonomischem Profil und einem Durchschnittsalter von 43,5 Jahren.

Das Relearning ermöglicht es Ihnen, mit weniger Aufwand und mehr Leistung zu lernen, sich mehr auf Ihr Fachgebiet einzulassen, einen kritischen Geist zu entwickeln, Argumente zu verteidigen und Meinungen zu kontrastieren: eine direkte Gleichung zum Erfolg.

Nach den neuesten wissenschaftlichen Erkenntnissen der Neurowissenschaften wissen wir nicht nur, wie wir Informationen, Ideen, Bilder und Erinnerungen organisieren, sondern auch, dass der Ort und der Kontext, in dem wir etwas gelernt haben, von grundlegender Bedeutung dafür sind, dass wir uns daran erinnern und es im Hippocampus speichern können, um es in unserem Langzeitgedächtnis zu behalten.

Auf diese Weise sind die verschiedenen Elemente unseres Programms im Rahmen des so genannten neurokognitiven kontextabhängigen E-Learnings mit dem Kontext verbunden, in dem der Teilnehmer seine berufliche Praxis entwickelt.

Dieses Programm bietet die besten Lehrmaterialien, die sorgfältig für Fachleute aufbereitet sind:

Studienmaterial

Alle didaktischen Inhalte werden von den Fachleuten, die den Kurs unterrichten werden, speziell für den Kurs erstellt, so dass die didaktische Entwicklung wirklich spezifisch und konkret ist.

Diese Inhalte werden dann auf das audiovisuelle Format angewendet, um die TECH-Online-Arbeitsmethode zu schaffen. Und das alles mit den neuesten Techniken, die dem Studenten qualitativ hochwertige Stücke aus jedem einzelnen Material zur Verfügung stellen.

Meisterklassen

Die Nützlichkeit der Expertenbeobachtung ist wissenschaftlich belegt.

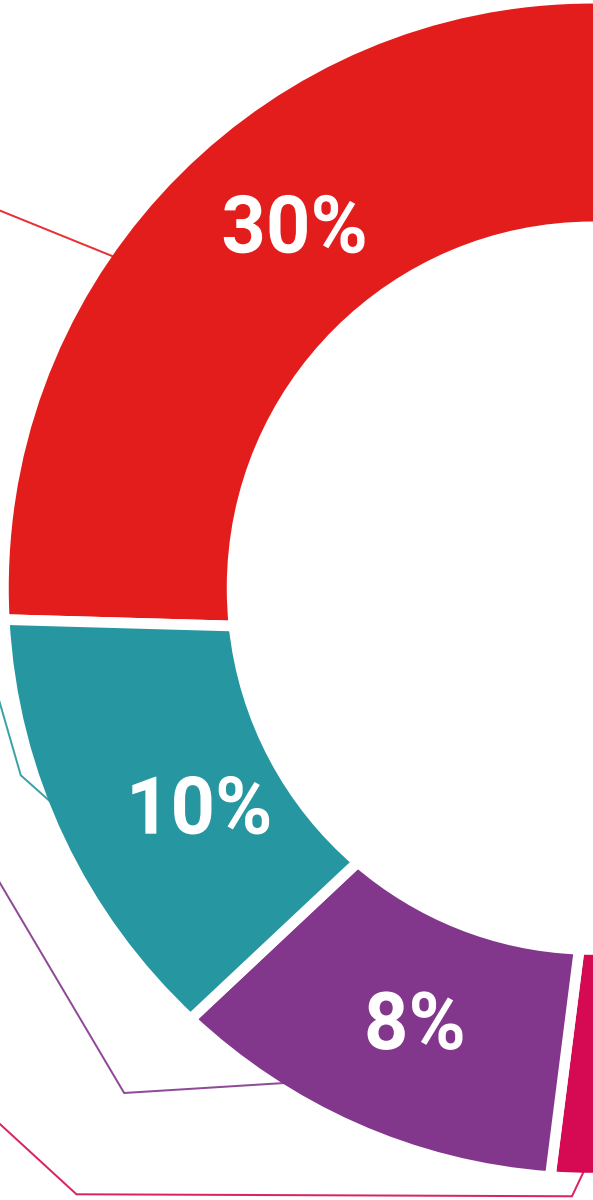
Das sogenannte Learning from an Expert baut Wissen und Gedächtnis auf und schafft Vertrauen für zukünftige schwierige Entscheidungen.

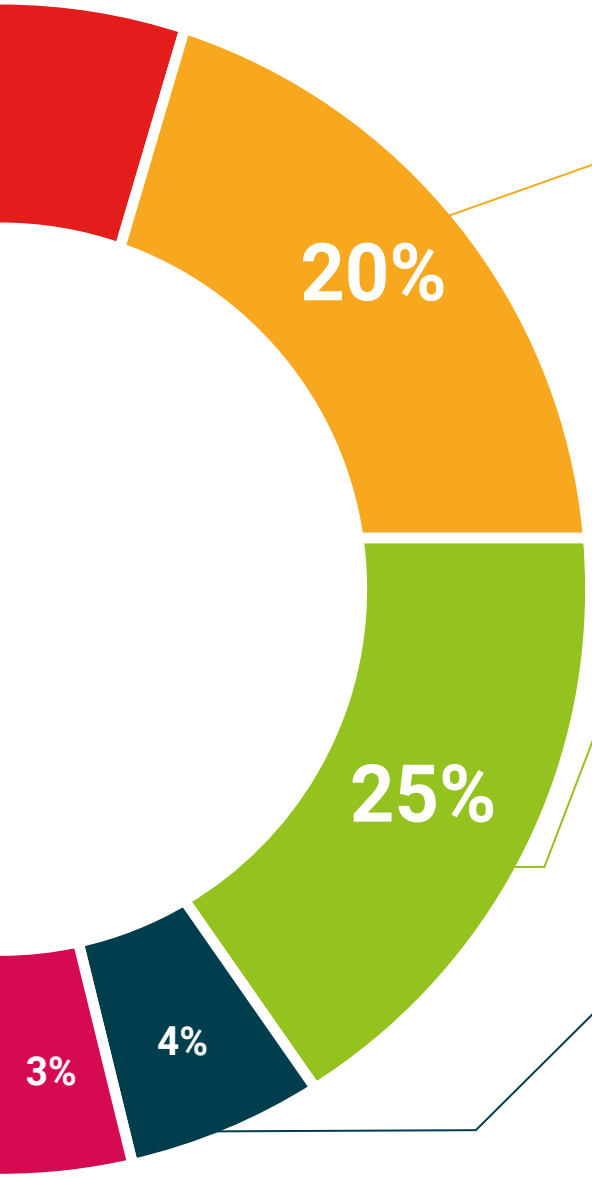
Fertigkeiten und Kompetenzen Praktiken

Sie werden Aktivitäten durchführen, um spezifische Kompetenzen und Fertigkeiten in jedem Fachbereich zu entwickeln. Praktiken und Dynamiken zum Erwerb und zur Entwicklung der Fähigkeiten und Fertigkeiten, die ein Spezialist im Rahmen der Globalisierung, in der wir leben, entwickeln muss.

Weitere Lektüren

Aktuelle Artikel, Konsensdokumente und internationale Leitfäden, u.a. In der virtuellen Bibliothek von TECH haben die Studenten Zugang zu allem, was sie für ihre Ausbildung benötigen.





Fallstudien

Sie werden eine Auswahl der besten Fallstudien vervollständigen, die speziell für diese Qualifizierung ausgewählt wurden. Die Fälle werden von den besten Spezialisten der internationalen Szene präsentiert, analysiert und betreut.

Interaktive Zusammenfassungen

Das TECH-Team präsentiert die Inhalte auf attraktive und dynamische Weise in multimedialen Pillen, die Audios, Videos, Bilder, Diagramme und konzeptionelle Karten enthalten, um das Wissen zu vertiefen.

Dieses einzigartige Bildungssystem für die Präsentation multimedialer Inhalte wurde von Microsoft als "europäische Erfolgsgeschichte" ausgezeichnet.

Prüfung und Nachprüfung

Die Kenntnisse der Studenten werden während des gesamten Programms regelmäßig durch Bewertungs- und Selbsteinschätzungsaktivitäten und -übungen beurteilt und neu bewertet, so dass die Studenten überprüfen können, wie sie ihre Ziele erreichen.

07

Qualifizierung

Der Privater Masterstudiengang in Informatik und Programmiersprachen garantiert neben der strengsten und aktuellsten Ausbildung auch den Zugang zu einem von der TECH Technologischen Universität ausgestellten Diplom.

Schließen Sie dieses Programm erfolgreich ab und erhalten Sie Ihren Universitätsabschluss, ohne lästige Reisen oder Formalitäten"

Dieser **Privater Masterstudiengang in Informatik und Programmiersprachen** enthält das vollständigste und aktuellste Programm auf dem Markt.

Sobald der Student die Prüfungen bestanden hat, erhält er/sie per Post* mit Empfangsbestätigung das entsprechende Diplom, ausgestellt von der **TECH Technologischen Universität**.

Das von **TECH Technologische Universität** ausgestellte Diplom drückt die erworbene Qualifikation aus und entspricht den Anforderungen, die in der Regel von Stellenbörsen, Auswahlprüfungen und Berufsbildungsausschüssen verlangt werden.

Titel: **Privater Masterstudiengang in Informatik und Programmiersprachen**

Anzahl der offiziellen Arbeitsstunden: **1.500 Std.**


tech technologische universität

Verleiht dieses

DIPLOM

an

Herr/Frau _____, mit Ausweis-Nr. _____

Für den erfolgreichen Abschluss und die Akkreditierung des Programms

PRIVATER MASTERSTUDIENGANG

in

Informatik und Programmiersprachen

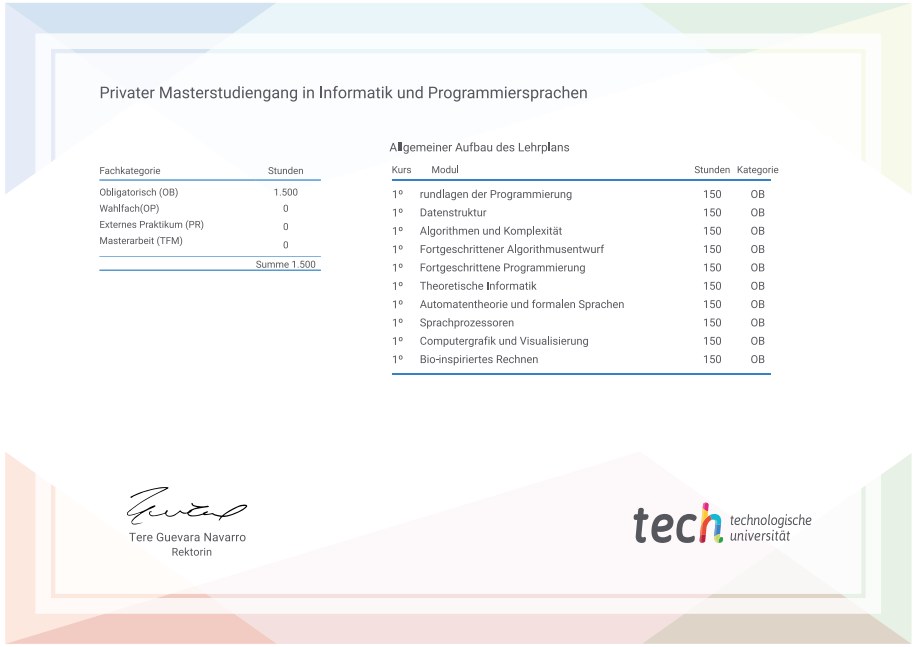
Es handelt sich um einen von dieser Universität verliehenen Abschluss, mit einer Dauer von 1.500 Stunden, mit Anfangsdatum tt/mm/jjjj und Enddatum tt/mm/jjjj.

TECH ist eine private Hochschuleinrichtung, die seit dem 28. Juni 2018 vom Ministerium für öffentliche Bildung anerkannt ist.

Zum 17. Juni 2020

Tere Guevara Navarro
Rektorin

Diese Qualifikation muss immer mit einem Hochschulabschluss einhergehen, der von der für die Berufsausübung zuständigen Behörde des jeweiligen Landes ausgestellt wurde. einzigartiger Code TECH: AFWOR23S techtitute.com/titel


Privater Masterstudiengang in Informatik und Programmiersprachen

Fachkategorie	Stunden
Obligatorisch (OB)	1.500
Wahlfach(OP)	0
Externes Praktikum (PR)	0
Masterarbeit (TFM)	0
	Summe 1.500

Allgemeiner Aufbau des Lehrplans

Kurs	Modul	Stunden	Kategorie
1°	rundlagen der Programmierung	150	OB
1°	Datenstruktur	150	OB
1°	Algorithmen und Komplexität	150	OB
1°	Fortgeschrittener Algorithmusentwurf	150	OB
1°	Fortgeschrittene Programmierung	150	OB
1°	Theoretische Informatik	150	OB
1°	Automatentheorie und formalen Sprachen	150	OB
1°	Sprachprozessoren	150	OB
1°	Computergrafik und Visualisierung	150	OB
1°	Bio-inspiriertes Rechnen	150	OB

Tere Guevara Navarro
Rektorin

tech technologische universität

*Haager Apostille. Für den Fall, dass der Student die Haager Apostille für sein Papierdiplom beantragt, wird TECH EDUCATION die notwendigen Vorkehrungen treffen, um diese gegen eine zusätzliche Gebühr zu beschaffen.

tech technologische universität

Privater Masterstudiengang
Informatik und Programmiersprachen

- Modalität: **online**
- Dauer: **12 Monate**
- Qualifizierung: **TECH Technologische Universität**
- Zeitplan: **in Ihrem eigenen Tempo**
- Prüfungen: **online**

Privater Masterstudiengang
Informatik und Programmiersprachen

tech technologische universität