



| Principali informazioni sull'insegnamento | |
|--|----------------|
| Denominazione dell'insegnamento | Programmazione |
| Corso di studio | Informatica |
| Anno di corso | Primo |
| Crediti formativi universitari (CFU) / European Credit Transfer and Accumulation System (ECTS):9+3 | : |
| SSD | INF/01 |
| Lingua di erogazione | Italiano |
| Periodo di erogazione | Primo semestre |
| Obbligo di frequenza | No |

| Docente | |
|--|---|
| Nome e cognome | Fabio Abbattista |
| Indirizzo mail | Fabio.abbattista@uniba.it |
| Telefono | |
| Sede | Dipartimento di informatica – Università' di Bari |
| Sede virtuale | |
| Ricevimento (giorni, orari e modalità) | Martedì 15.00-16.00 |

| Syllabus | |
|--|--|
| Obiettivi formativi | <ul style="list-style-type: none">• Conoscenza e capacità di comprensione Lo studente dovrà essere in grado di analizzare e risolvere semplici problemi, progettando e sviluppando programmi in un linguaggio di programmazione di alto livello.• Conoscenza e capacità di comprensione applicate Lo studente dovrà acquisire competenze relative a:<ul style="list-style-type: none">- Traduzione di semplici algoritmi in programmi correttamente funzionanti;- Capacità di individuazione di malfunzionamenti attraverso il debugging;- Capacità di problem-solving attraverso l'applicazione di nozioni apprese nelle discipline informatiche di base nella pratica della programmazione.• Autonomia di giudizio Lo studente deve dimostrare di aver acquisito autonomia di giudizio e capacità di valutazione degli algoritmi sviluppati da lui o da terzi.• Capacità di apprendere Lo studente dovrà mostrare di essere in grado di orientarsi agevolmente nelle problematiche relative alla comprensione e all'utilizzo delle tecnologie e dei metodi di competenza per lo sviluppo di algoritmi e per la loro traduzione in programmi per computer. |
| Prerequisiti | Buona comprensione della lingua inglese. |
| Contenuti di insegnamento (Programma) | 1.Introduzione Problem solving: algoritmi e programmi. |



| | |
|--|--|
| | <p>Programmazione: una definizione Linguaggi assemblativi e di alto livello. Linguaggi imperativi. Cenni sulla loro evoluzione. Linguaggi e grammatiche. Sintassi dei linguaggi di programmazione. Diagrammi sintattici. Forma di Backus-Naur. Cenni sui compilatori.</p> <p>2. Problem solving Definizione e proprietà degli algoritmi. Alcuni semplici esempi. Progettazione di un algoritmo per raffinamenti successivi. Astrazione</p> <p>3. Rappresentazione di algoritmi Specifiche di un algoritmo: diagrammi di flusso, albero di decomposizione, linguaggio naturale, pseudocodice. Costrutti base: la sequenza. Esempi. Costrutti base: la selezione. Esempi. Costrutti base: la iterazione. Esempi. Programmazione strutturata. Teorema di Bohem-Jacopini (enunciato). Algoritmi elementari: conteggio, sommatoria di un insieme di numeri, calcolo del fattoriale, massimo comun divisore</p> <p>4. Progettazione del software Cenni su programmazione in grande e programmazione in piccolo e sulle metodologie di progetto top-down e bottom-up. Analisi dei requisiti. Esempi. Progetto del software. Esempi. Codifica e debug. Correttezza, classificazione degli errori. Esempi. Test di un programma: metodi basati sulle specifiche. Esempi.</p> <p>5. Linguaggi di programmazione: dati predefiniti Tipi di dato. Tipi semplici. Compatibilità ed equivalenza tra tipi di dato. Variabili e costanti. Istruzione di assegnazione. Dati strutturati: array. Dati strutturati: stringhe. Esempi Algoritmi su array e stringhe: ricerca del massimo e minimo di istogrammi mediante array, inversione degli elementi e rimozione valori duplicati. Algoritmi su matrici: somma e prodotto di 2 matrici, trasposta di una matrice quadrata.</p> <p>6. Linguaggi di programmazione: controllo Strutture di controllo di base. Astrazione funzionale mediante sottoprogrammi (procedure e funzioni). Identificatori e scope di un identificatore. Valore di ritorno delle funzioni. Esempi Parametri formali ed effettivi. Esempi. Tecniche di legame dei parametri: per valore. Esempi.</p> |
|--|--|



| | |
|-------------------------------------|--|
| | <p>Tecniche di legame dei parametri: per indirizzo. Esempi.</p> <p>7. I file Definizione di file. Tipi di file. Gestione dei file di testo. Esempi. Gestione dei file binari. Esempi.</p> <p>8. Algoritmi fondamentali Algoritmo di ricerca binaria. Algoritmi di ordinamento: ordinamento per selezione, per inserzione e per scambi.</p> |
| Testi di riferimento | <p>P. Deitel e H. Deitel, Il Linguaggio C - Fondamenti e tecniche di programmazione, Pearson, 2013. A. Downey, Pensare in Python – Come pensare da informatico, O'Reilly, 2018</p> |
| Note ai testi di riferimento | <p>Testi integrativi: B.W. Kernighan e R. Pike, Programmazione nella pratica, Addison-Wesley, 1999. J.R. Hanly, E.B. Koffman, Problem solving e programmazione in C, Apogeo, 2013</p> |

| | | | |
|---------------------------------------|--------------------|--|--------------------|
| Organizzazione della didattica | | | |
| Ore | | | |
| Totali | Didattica frontale | Pratica (laboratorio, campo, esercitazione, altro) | Studio individuale |
| 300 | 72 | 45 | 183 |
| CFU/ETCS | | | |
| 12 | 9 | 3 | |

| | |
|-------------------------|-------------------|
| Metodi didattici | Learning by doing |
| | |

| | |
|--|--|
| Risultati di apprendimento previsti | |
| Conoscenza e capacità di comprensione | Lo studente dovrà essere in grado di analizzare e risolvere semplici problemi, progettando e sviluppando programmi in un linguaggio di programmazione di alto livello. |
| Conoscenza e capacità di comprensione applicate | <p>Lo studente dovrà acquisire competenze relative a:</p> <ul style="list-style-type: none"> - Traduzione di semplici algoritmi in programmi correttamente funzionanti e ben documentati; - Capacità di individuazione di malfunzionamenti attraverso il debugging; - Capacità di problem-solving attraverso l'applicazione di nozioni apprese nelle discipline informatiche di base nella pratica della programmazione. |
| Competenze trasversali | <ul style="list-style-type: none"> • Autonomia di giudizio Lo studente deve dimostrare di aver acquisito autonomia di giudizio e capacità di valutazione degli algoritmi sviluppati da lui o da terzi. • Capacità di apprendere Lo studente dovrà mostrare di essere in grado di orientarsi agevolmente nelle problematiche relative alla comprensione e all'utilizzo delle tecnologie e dei metodi di competenza per lo sviluppo di algoritmi e per la loro traduzione in programmi per computer. |



| | |
|---|--|
| Valutazione | |
| Modalità di verifica dell'apprendimento | Alcune prove pratiche da svolgere in itinere, non obbligatorie. Il superamento delle prove in itinere e/o i risultati delle esercitazioni pratiche attribuiscono una premialità sul voto finale. L'esame consiste di una prova pratica e in una prova orale individuale. |
| Criteri di valutazione | Lo studente dovrà dimostrare di aver acquisito la capacità di progettare l'algoritmo ottimale per la soluzione di problemi con caratteristiche diverse. Inoltre deve aver sviluppato buone competenze nell'utilizzo di un linguaggio di programmazione di alto livello. L'esame finale consiste nella risoluzione di un problema, identificando l'algoritmo e sviluppando il corrispondente programma (per chi non è stato esonerato) e in una prova orale. |
| Criteri di misurazione dell'apprendimento e di attribuzione del voto finale | |
| Altro | |
| | |



| General information | |
|---|---------------------------------------|
| Academic subject | Computer Programming |
| Degree course | Computer Science |
| Academic Year | First |
| European Credit Transfer and Accumulation System (ECTS) | 9+3 |
| Language | Italian |
| Academic calendar (starting and ending date) | Septembre 26, 2022 – January 13, 2023 |
| Attendance | Not mandatory |

| Professor/ Lecturer | |
|-------------------------|--|
| Name and Surname | Fabio Abbattista |
| E-mail | Fabio.abbattista@uniba.it |
| Telephone | |
| Department and address | Dipartimento di Informatica, Università' di Bari |
| Virtual headquarters | |
| Tutoring (time and day) | Tuesday, 15.00-16.00 |

| Syllabus | |
|-----------------------------|--|
| Learning Objectives | <ul style="list-style-type: none">• Knowledge and understanding The student should be able to analyze and solve simple problems, designing and developing programs in an high-level programming language.• Applied knowledge and understanding The student will have to acquire skills related to:<ul style="list-style-type: none">- Translation of simple algorithms into properly functioning programs;- Ability to identify malfunctions through debugging;- Problem-solving skills through the application of notions learned in basic computer disciplines in the practice of programming.• Autonomy of judgment The student must demonstrate that he has acquired autonomy of judgment and evaluation skills in the context of the development of algorithms.• Ability to learn The student must show that they have developed the ability to learn and to orient themselves easily in the problems related to the understanding and use of information technologies in its specific field of application. |
| Course prerequisites | Good understanding of the English language. |
| Contents | <p>1. Introduction Problem solving: algorithms and computer programs. A definition of computer programming. Assembly and high level programming languages. Evolution of imperative programming languages. Languages and grammars. Computer programming syntax. Syntactic diagrams. Backus-Naur Form. Compilers.</p> <p>2. Problem solving Definition and properties of algorithms.</p> |



| | |
|--|---|
| | <p>Simple examples. Designing an algorithm: stepwise-refinements. Abstraction</p> <p>3. Algorithms representation Flow-chart, decomposition tree, natural language, pseudocode. Basic structure: sequence. Examples. Basic structure: selection. Examples. Basic structure: iteration. Examples. Structured programming. Bohem-Jacopini theorem. Simple algorithms: count, sum of numbers, factorial, greatest common divisor.</p> <p>4. Software design Programming in the large and programming in the small. Top-down and bottom-up design. Requirements analysis. Examples. Software design. Examples. Coding and debugging. Correctness, Error classification. Examples. Software testing based on specifications. Examples.</p> <p>5. Programming languages: data type Simple data types. Compatibility and equivalence. Variables and constants. Assignment statement. Structured data type: array. Structured data type: strings. Array and strings algorithms: search for the maximum and the minimum, calculation of the average value, inversion of the elements of an array, removing duplicate elements of an array. Matrices algorithms: matrix sum and multiplication, smatrix transpose.</p> <p>6. Programming languages: control Basic control structures. Functional abstraction: procedures and functions. Identifier, scope of an identifier. Return value of functions. Examples. Formal and actual parameters. Examples. Parameter binding: by value. Examples. Parameter binding: by reference. Examples.</p> <p>7. Files File definition. File types. Text files management. Examples. Binary files management. Examples.</p> <p>8. Fundamental algorithms Binary search algorithm. Sorting algorithms: selection, insertion, bubble.</p> |
|--|---|



| | |
|-------------------------------|--|
| Books and bibliography | P. Deitel e H. Deitel, Il Linguaggio C - Fondamenti e tecniche di programmazione, Pearson, 2013. A. Downey, Pensare in Python – Come pensare da informatico, O’Reilly, 2018 |
| Additional materials | B.W. Kernighan e R. Pike, Programmazione nella pratica, Addison-Wesley, 1999. J.R. Hanly, E.B. Koffman, Problem solving e programmazione in C, Apogeo, 2013 |

| | | | |
|---|----------|---|---|
| Work schedule | | | |
| Total | Lectures | Hands on (Laboratory, working groups, seminars, field trips) | Out-of-class study hours/ Self-study hours |
| Hours | | | |
| 300 | 72 | 45 | 183 |
| ECTS | | | |
| 12 | 9 | 3 | |
| Teaching strategy | | Learning by doing | |
| Expected learning outcomes | | | |
| Knowledge and understanding on: | | The student should be able to analyze and solve simple problems, designing and developing programs in an high-level programming language. | |
| Applying knowledge and understanding on: | | The student will have to acquire skills related to: - Translation of simple algorithms into properly functioning programs; - Ability to identify malfunctions through debugging; - Problem-solving skills through the application of notions learned in basic computer disciplines in the practice of programming. | |
| Soft skills | | <ul style="list-style-type: none"> • Autonomy of judgment The student must demonstrate that he has acquired autonomy of judgment and evaluation skills in the context of the development of algorithms. • Ability to learn The student must show that they have developed the ability to learn and to orient themselves easily in the problems related to the understanding and use of information technologies in its specific field of application. | |

| | |
|---|---|
| Assessment and feedback | |
| Methods of assessment | Some practical tests are held during the semester. The grades obtained in the tests award a bonus on the final grade. The final exam consists of an individual practical test. |
| Evaluation criteria | The student will have to demonstrate that he has acquired the ability to design an algorithm for solving problems with different characteristics. In addition, he must have developed good skills in the use of an high-level programming language. <ul style="list-style-type: none"> • The final exam consists in solving a problem by identifying the algorithm and developing the related program (for those who have not taken / passed the exemption tests) and in the oral. |
| Criteria for assessment and attribution of the final mark | |
| Additional information | |
| | |