



Novice Developer to Pythonista

Głównym wyzwaniem przed którym stają dziś organizacje na całym świecie jest konieczność ciągłego podnoszenia umiejętności i poziomu wiedzy w ślad za gwałtownym rozwojem nowych technologii i zmian na globalnym rynku.

Stały rozwój i podnoszenie kwalifikacji w IT od dawna jest już rzeczą oczywistą, a możliwość zapewnienia wsparcia specjalistom chcącym stale się rozwijać jest jedną z głównych kart przetargowych w walce o pracownika.

Na rynku liczą się dziś ludzie, którzy posiadają konkretne kompetencje i zestaw umiejętności pozwalający im wykonywać zadania efektywnie, a nie Ci z najdłuższym stażem pracy.

Dziś, bardziej niż kiedykolwiek w cenie jest umiejętność budowania ścieżki kariery dla profesjonalistów IT, którzy wciąż chcą się liczyć na rynku pracy.

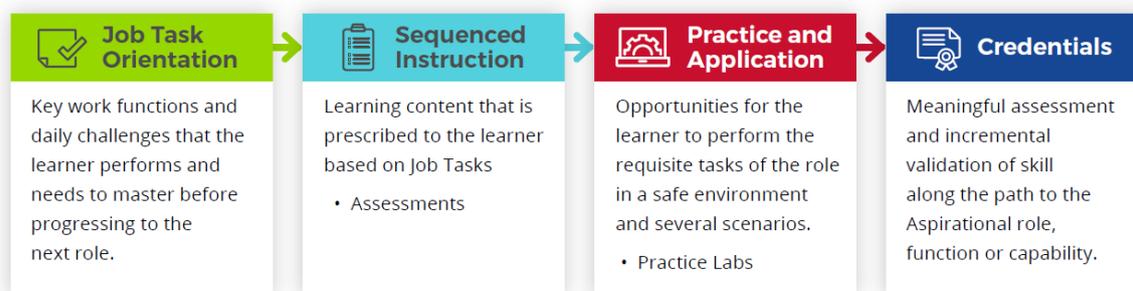
Skillsoft Aspire Journey stanowi odpowiedź na pytanie, jakie szkolenia muszą ukończyć, aby być przygotowanym do mojej wymarzonej pracy. Spośród kilkuset kanałów tematycznych dostępnych na naszej platformie szkoleniowej nasi specjaliści wybrali te, które naszym zdaniem najlepiej wyposażą uczących się w narzędzia potrzebne do realizacji zadań w nowej roli.

Skillsoft Aspire Journey to zestawy szkoleń i ćwiczeń w języku angielskim, które metodycznie, krok po kroku pozwalają specjalistom przejść od poziomu podstawowego do zaawansowanego.

Każda ścieżka zawiera szkolenia, laboratoria wirtualne, video i książki, które pomogą uczącym się osiągnąć pożądane kompetencje poświadczone certyfikatem.

Aspire Journey Model

Cała ścieżka opiera się na 4-elementowym cyklu powtarzanym na kolejnych etapach nauki.

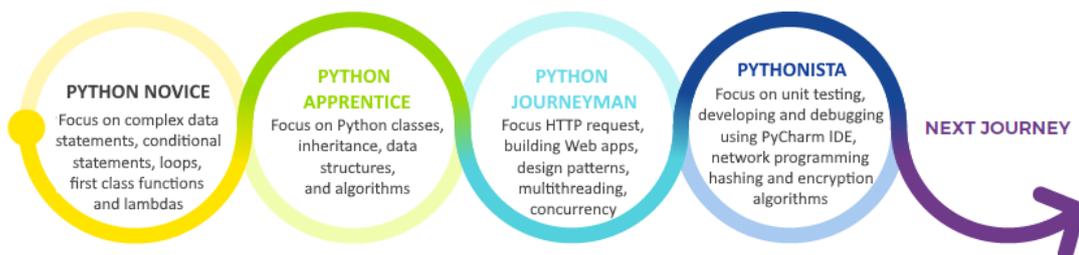


1. Określenie kluczowych funkcji i wyzwań, z którymi musi poradzić sobie uczący się w chwili obecnej, jak i tymi, z którymi przyjdzie mu się zmierzyć w nowej pracy.
2. Przejście zaprojektowanych ścieżek w proponowanej kolejności, wykonanie ćwiczeń i zaliczenie testów.
3. Przećwiczenie nowych umiejętności w kontrolowanym środowisku w oparciu o gotowe scenariusze działań. Laboratoria wirtualne Skillsoft
4. Certyfikat – zaliczenie testu końcowego na poziomie co najmniej 70% i uzyskanie certyfikatu potwierdzającego ukończenie danego etapu nauki.

Aspire Journey – Novice Developer to Pythonista

Analizując trendy opisujące zachowanie użytkowników na naszych platformach szkoleniowych i współpracując ściśle z naszymi klientami na całym świecie Skillsoft wyselekcjonował najlepsze materiały szkoleniowe i ułożył je w ustrukturalizowaną ścieżkę rozwoju. W zależności od tematu ścieżka zawiera od około 30 do 90 godzin szkoleń.

PYTHON NOVICE TO PYTHONISTA



Pythonista Track 1 – Python Novice (Advance Release)

In this Skillsoft Aspire track of the Pythonista journey, the focus is getting started with Python, complex data types, conditional statements and too...

- starting Python
- complex data types conditional statements and loops
- and first class functions and lambdas

duration:
14h 56m 32s



Pythonista Track 2 – Python Apprentice (Advance Release)

In this Skillsoft Aspire track of the Pythonista journey, the focus is Python classes and inheritance and also data structures and algorithms.

- Python classes
- Inheritance
- data structures
- algorithms

duration:
17h 21m 17s



Pythonista Track 3 – Python Journeyman (Advance Release)

In this Skillsoft Aspire track of the Pythonista journey, the focus is working with HTTP requests, building Web apps, design patterns, and multi...

- working with HTTP requests,
- building Web apps,
- design patterns
- multithreading
- concurrency

duration:
16h 17m 24s



Pythonista Track 4 – Pythonista (Advance Release)

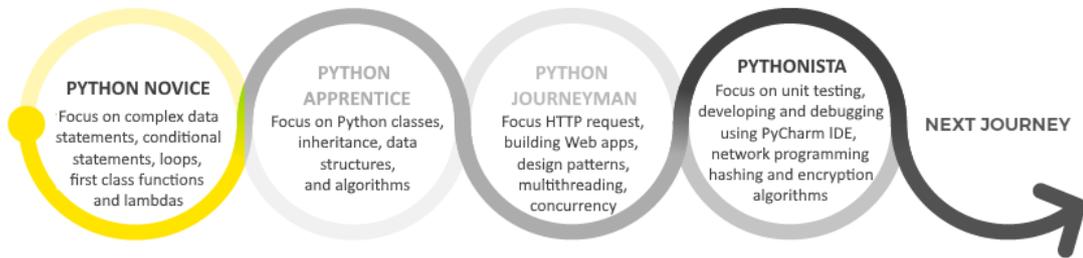
In this Skillsoft Aspire track of the Pythonista journey, the focus is unit testing, developing and debugging using the PyCharm IDE, wrangling Ex...

- unit testing, developing and debugging using the PyCharm IDE
- wrangling Excel data
- network programming, and hashing and encryption algorithms

duration:
6h 13m 36s

Track 1: Python Novice (duration: 14h 56m 32s)

PYTHON NOVICE TO PYTHONISTA



Getting Started with Python: Introduction



Complex Data Types in Python: Working with Lists & Tuples in Python

objectives:

- install the Anaconda distribution of Python to run Python in a Windows environment
- run the Jupyter notebook server to execute Python code on a Windows machine
- install the Anaconda distribution of Python to run Python on a Mac machine
- run the Jupyter notebook server to execute Python code on a Mac machine
- execute Python code to perform simple calculations
- use built-in functions in Python to perform operations
- use variables in place of values directly
- create floating point, integer, Boolean, and string variables
- assign values to newly created variables
- perform operations to update variable values
- work with floats, integers, Booleans, and strings
- initialize and work with single-line and multi-line strings
- perform string formatting operations to display data
- recall features of Jupyter notebooks, built-in functions, and variables

objectives:

- create and initialize lists in Python
- access and update list elements
- add, remove, sort, and reverse elements from a list
- executing built-in functions with lists
- create new lists from existing lists using slicing operations
- extract specific elements from the original list using step size
- perform list functions on strings
- invoke functions on the string object
- access substrings using slicing operations
- specify the similarities between lists and tuples
- specify the differences between lists and tuples
- use dictionaries and sets in Python
- recall the ways in which lists and tuples are similar and different



Complex Data Types in Python: Working with Dictionaries & Sets in Python

objectives:

- create and initialize dictionaries
- create dictionaries with other complex data types as values
- modify and update dictionaries using dictionary methods
- create and initialize sets
- perform union, intersection, difference, and other set operations
- work with nested types within other complex data types
- convert lists to dictionaries and vice versa
- recall feature of dictionaries and sets



Complex Data Types in Python: Shallow & Deep Copies in Python

objectives:

- create and initialize lists in Python
- access and update list elements
- add, remove, sort, and reverse elements from a list
- executing built-in functions with lists
- create new lists from existing lists using slicing operations
- extract specific elements from the original list using step size
- perform list functions on strings
- invoke functions on the string object
- access substrings using slicing operations
- specify the similarities between lists and tuples
- specify the differences between lists and tuples
- use dictionaries and sets in Python
- recall the ways in which lists and tuples are similar and different



Conditional Statements & Loops: If-else Control Structures in Python

objectives:

- identify how conditions in Python work
- evaluate conditions involving primitive data types using if statements
- evaluate conditions involving complex data types using if statements
- evaluate multiple conditions for the purpose of decision making
- evaluate multiple conditions for the purpose of decision making using the nested control structures
- identify how to use the if-else statement to make decisions involving complex data types such as lists, tuples, and dictionaries
- recall how to convert an integer to a float, and a float or an integer to a string and vice-versa
- recall how to convert a primitive datatype to a complex datatype and to convert between various complex datatypes
- recall how to convert between various complex datatypes and view base conversions using the Python built-in functions
- implement basic concepts into some real programming such as conversions between datatypes and some Python built-in methods
- solve various programming problems using some Python built-in methods
- solve various programming problems using if-elif-else statements and nested if-else statements
- solve coding problems using a combination of the different forms of if-else statements
- recall details related to the order of precedence in Python and apply if-else statements to solve basic programming problems



Conditional Statements & Loops: The Basics of for Loops in Python

objectives:

- use for loops to process the elements in a list and the characters in a string
- code for loops to iterate over values in a tuple and the keys and values in a dictionary
- recognize the function of associating an else block with a Python for loop
- include if-else statements and other for loops within a for loop
- generate a sequence of consecutive integers with the range function
- define the interval in a sequence of increasing and decreasing integers using the range function
- use the range function to iterate over a large range of values and apply it within nested for loops
- write for loops in order to iterate over 1-dimensional and 2-dimensional sequences



objectives:

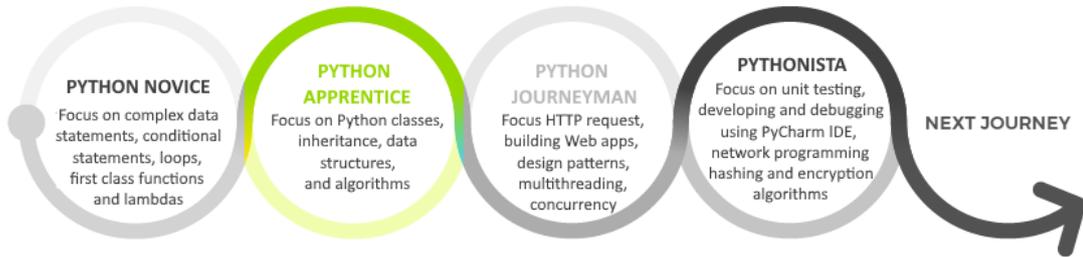
- terminate a for loop when a specific condition is met using the break statement
- recognize how the break statement affects the code in the else block of a for loop
- skip an iteration of a for loop when a specific condition is met using the continue statement
- use the continue statement along with the break statement within the same for loop
- convey the fact that no action is performed under specific conditions by using the pass statement
- create a list out of the contents of another list using a comprehension
- specify conditions in list comprehensions in order to filter elements used in the source list and to define the values in the newly created list
- write for loops that make use of the break and continue statements to control flow and use comprehensions to generate a list

objectives:

- implement a basic while loop and recognize what conditions can cause it to become an infinite loop
- use while loops to carry out actions while evaluating expressions based on numerical and string data
- define while loops whose iterations depend on user input data
- recall the syntax for defining while loops within a single line
- iterate over a list of elements using while loops
- iterate over multiple lists and tuples using while loops
- identify when it is appropriate to use the break keyword to stop a while loop
- use the break statement in multiple scenarios to break out of a while loop and recognize the use of the pass keyword within such loops
- skip steps in individual iterations of a while loop using the continue statement
- compare while loops and for loops and implement a while loop which terminates only when the user enters a specific input

Track 2: Python Apprentice (duration: 17h 21m 17s)

PYTHON NOVICE TO PYTHONISTA



Advanced Python Topics:
File Operations in Python



Advanced Python Topics:
Exceptions & Command
Line Arguments

objectives:

- use the open function in Python to open a file for reading
- recognize the differences between the read(), readline(), and readlines() functions when working with files in Python
- recall the differences between opening a file in write mode and append mode
- distinguish between the r+ and a+ modes to read from and write to a file
- use the load and loads functions of the json module to parse JSON data
- convert Python dictionaries and lists into JSON strings and files
- identify some of the file formats that can be handled by the CSV module in Python
- convert Python dictionaries and lists into CSV files
- define a customized file format creating a CSV dialect and use that to parse and write data
- summarize the key concepts covered in this course

objectives:

- use a try and except block to handle a Python exception
- control how you handle exceptions that your code may raise
- recognize how exceptions are defined in a hierarchy and how related exceptions can be caught and handled
- define multiple except blocks to handle various exceptions that can be raised by your code
- convert code prototyped in a Jupyter notebook into a Python script that can be executed from a shell
- run your code from the Python shell and recognize how to execute single-line and multi-line commands
- use the sys and argparse module to access command line arguments to a Python script
- parse and use the arguments passed to a Python script from the command line
- define the names and other customizable features of command line arguments to a Python script using the argparse module
- summarize the key concepts covered in this course



Advanced Python Topics: Modules & Virtual Environments

objectives:

- use pre-built Python modules to perform common operations by importing them into your source code
- recognize the features available in popular Python libraries such as NumPy, random, and datetime
- package Python code into a tar archive for distribution to end users
- use pip to install a module in your environment and import it into your Python applications
- install the virtualenv tool and use it to create a virtual environment for Python applications
- activate and run Python scripts from within virtual environments
- install different packages in different virtual environments and recognize how they are isolated from each other
- summarize the key concepts covered in this course



Advanced Python Topics: Migrating from Python 2 to Python 3

objectives:

- use pre-built Python modules to perform common operations by importing them into your source code
- recognize the features available in popular Python libraries such as NumPy, random, and datetime
- package Python code into a tar archive for distribution to end users
- use pip to install a module in your environment and import it into your Python applications
- install the virtualenv tool and use it to create a virtual environment for Python applications
- activate and run Python scripts from within virtual environments
- install different packages in different virtual environments and recognize how they are isolated from each other
- summarize the key concepts covered in this course



objectives:

- recall how state and behavior can be encapsulated in a single unit
- describe how classes can be used as blueprints to create objects
- compare objects and instances to classes
- model is-a relationships using inheritance
- describe the advantages of using object-oriented programming
- describe classes, define how state and behavior of a class are represented, list the characteristics of class objects or instances, describe class inheritance, and list the advantages of object-oriented programming with classes

objectives:

- create a classes using Python
- assign attributes to objects of classes
- initialize class variables using the init special method
- pass arguments to initialize the state of a class object
- define additional methods in a class
- recall how class variables work
- recall how class variables are different from instance variables
- recall how class variables share memory across objects of a class
- work with variables that have their own memory in each object
- define getters and setters for each instance variable
- prevent inadvertent modification of instance variables
- create a class to represent a real-world entity
- parse information to create classes using a dictionary
- describe the use of the init method in a class, specify why the self argument is passed to methods in a class, differentiate between class and instance variables, and specify how member variables can be made private



Python Classes & Inheritance: Working with Inheritance in Python



Python Classes & Inheritance: Advanced Functionality Using Python Classes

objectives:

- recall the default base class for all Python classes
- model an is-a relationship using inheritance
- invoke base class methods from subclasses
- provide implementations for base class methods
- work with class hierarchies
- define methods in a subclass
- define multiple inheritance levels in classes
- define multiple base classes for a single subclass
- recall what polymorphism is
- implement polymorphism in Python
- implement base and derived classes, specify an init method to initialize member variables, define getters and setters for member variables, and override a method

objectives

- represent objects using customized strings
- perform addition operations on custom objects
- perform subtraction operations on custom objects
- perform multiplication operations on custom objects
- perform floor division, modulo, and power-of operations
- allow built-in functions to work with custom data types
- execute for-loops on custom data types
- define properties on classes for intuitive use
- define properties using a simpler syntax
- work with class methods to access and update class state
- work with utility methods on classes
- define classes as abstract
- list special methods and what they represent, define a class and create a property within it, and differentiate between class methods and static methods



objectives:

- identify what makes a data structure and some of the purposes they serve
- recall the metrics on which algorithms and operations on data are evaluated
- recognize how the performance of operations and algorithms is expressed in terms of the size of the input
- describe a linked list, and its contents and structure
- summarize the different ways in which nodes can be added to a linked list and how search operations work on this data structure
- recall different methods to remove nodes from a linked list and describe the process of reversing the order of nodes in this data structure
- describe techniques used to keep track of the number of elements in a linked list
- summarize the workings of a stack data structure, including the addition and removal of elements
- identify some of the operations on stacks, such as ISEMPTY and ISFULL, and recall the complexities of the different stack operations
- describe the queue data structure and compare it to stacks
- summarize the time complexities of the common operations on linked lists and compare the stack and queue data structures

objectives:

- identify operations that run in constant time regardless of input
- recognize code whose time complexity varies directly with the value of the input
- identify tasks whose time complexity varies linearly with the size of the input
- recognize operations whose time complexity varies as the square of the input size
- use the native Queue class of Python and perform the standard queue operations on it
- code a bespoke Queue class that includes definitions for many of the standard queue operations, such as enqueue and dequeue
- recognize how a Python list can be used as a stack by loading and unloading elements from the same end
- implement a custom Stack class that includes functions for the common stack operations
- define a Linked List class and implement functions to insert a node at the head or the tail of the linked list
- code functions to perform search and delete operations in a linked list and to reverse the ordering of its nodes
- instantiate a Linked List and test out the various operations that have been defined
- summarize common operations on stacks and their time complexities, and list the situations when adding to or removing elements from a stack or queue can throw exceptions



objectives:

- identify the various properties of sorting algorithms that must be considered when selecting the right one for your data
- describe the operations involved when sorting a list of values using the Selection Sort algorithm
- summarize the process of a Bubble Sort when applied to a list of values
- recall the performance of the Bubble Sort on various measures such as time, space, and number of swaps
- describe the steps involved in performing an Insertion Sort and compare it to the Bubble Sort
- outline the workings of the Shell Sort and recall the performance metrics of this divide and conquer algorithm
- describe the process of sorting a list of elements using Merge Sort and list the complexity of this algorithm on various measures
- describe how the Quicksort algorithm partitions and sorts a list of elements
- recall the performance metrics of the Quicksort algorithm
- describe considerations for picking a sorting algorithm, list the properties of the Insertion Sort algorithm, and summarize the Shell Sort algorithm

objectives:

- write the code to implement a Selection Sort
- implement the Bubble Sort algorithm in Python
- code a function to implement the Insertion Sort algorithm
- write the code to implement the divide-and-conquer Shell Sort algorithm
- invoke the Shell Sort algorithm on an array of integers and examine the output at each iteration to understand how it works
- code a function to implement the Merge Sort algorithm and test it on an array of integers
- write the partition and Quicksort functions in order to implement a Quicksort
- apply Quicksort on an array of integers and analyze the results at each iteration to understand how the algorithm works
- describe the Bubble Sort algorithm and summarize the two functions needed to implement the Quicksort algorithm



objectives:

- identify the various properties of sorting algorithms that must be considered when selecting the right one for your data
- describe the operations involved when sorting a list of values using the Selection Sort algorithm
- summarize the process of a Bubble Sort when applied to a list of values
- recall the performance of the Bubble Sort on various measures such as time, space, and number of swaps
- describe the steps involved in performing an Insertion Sort and compare it to the Bubble Sort
- outline the workings of the Shell Sort and recall the performance metrics of this divide and conquer algorithm
- describe the process of sorting a list of elements using Merge Sort and list the complexity of this algorithm on various measures
- describe how the Quicksort algorithm partitions and sorts a list of elements
- recall the performance metrics of the Quicksort algorithm
- describe considerations for picking a sorting algorithm, list the properties of the Insertion Sort algorithm, and summarize the Shell Sort algorithm

objectives:

- write the code to implement a Selection Sort
- implement the Bubble Sort algorithm in Python
- code a function to implement the Insertion Sort algorithm
- write the code to implement the divide-and-conquer Shell Sort algorithm
- invoke the Shell Sort algorithm on an array of integers and examine the output at each iteration to understand how it works
- code a function to implement the Merge Sort algorithm and test it on an array of integers
- write the partition and Quicksort functions in order to implement a Quicksort
- apply Quicksort on an array of integers and analyze the results at each iteration to understand how the algorithm works
- describe the Bubble Sort algorithm and summarize the two functions needed to implement the Quicksort algorithm



objectives:

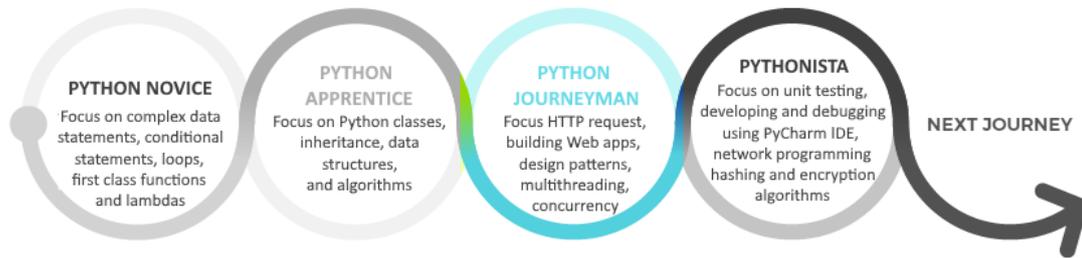
- describe how a sorted list of elements can be searched efficiently using a binary search
- recognize what trees and binary trees are and recall the properties of a binary search tree
- summarize how insert and lookup operations occur in a BST
- identify the minimum and maximum values in a BST, identify the greatest depth of the data structure, and calculate the sum of values from the root to a leaf node
- recall the different ways in which to traverse a BST and describe the method to perform a breadth first traversal
- summarize the pre-order and in-order depth first traversal techniques for a BST
- describe the post-order traversal technique for a BST
- identify the components that make up a graph and recognize the different terms associated with these data structures
- recognize the different ways to represent graphs and describe the structure of an adjacency matrix
- summarize the representation of a graph in the form of an adjacency list and adjacency set
- traverse over the nodes in a graph using the topological sort algorithm
- summarize the properties of a binary search tree and list three different ways in which a graph can be represented

objectives:

- code a function to perform a binary search on a sorted array of elements
- define the classes and functions required to implement a binary search tree
- create functions to perform common BST operations such as lookup and finding the minimum and maximum values
- write a function to perform a breadth first traversal of a BST
- code functions to perform pre-order, in-order, and post-order traversals of a BST
- define an abstract base class for a graph implementation and a vertex class with an adjacency set
- implement an adjacency set representation for a graph
- build a graph represented as an adjacency set and test out the functions defined to work with it
- define a class to represent a graph in the form of an adjacency matrix
- code a function to perform a breadth first traversal of a graph
- write a function to traverse a graph in a depth first manner
- implement a topological sort of a directed acyclic graph
- list the three different forms of depth first traversal of a binary tree and describe the Topological Sort algorithm for a graph

Track 3: Python Journeyman (duration: 16h 17m 24s)

PYTHON NOVICE TO PYTHONISTA



Python Unit Testing: An Introduction to Python's unittest Framework



Python Unit Testing: Advanced Python Testing Using the unittest Framework

objectives:

- install the latest version of Python and write a test using the unittest framework
- define and execute multiple tests within a single test case
- recognize the effect of the test names on the execution of test methods
- pick individual tests from a script which need to be executed
- identify the various assert functions available in unittest and their specific use cases
- use the different decorators available in unittest to conditionally and unconditionally skip specific tests
- summarize the key concepts covered in this course

objectives:

- recognize when a test script will benefit from the use of fixture functions
- use the setUp and tearDown functions to define common operations for tests in a script
- apply the setUpClass and tearDownClass functions in a Python test script
- group test cases into a test suite and execute them using a test runner
- use the makeSuite function to initialize a test suite
- download, install and configure the PyCharm IDE
- create a script which uses the unittest framework by using the PyCharm IDE
- define a script testing out multiple functions in the source using the PyCharm IDE
- summarize the key concepts covered in this course



Python Unit Testing: Testing Python Code Using pytest

objectives:

- write and execute a test using pytest
- execute tests defined in multiple scripts using pytest
- pick the tests to execute based on the test function names
- stop the execution of tests when there are a specific number of test failures
- apply markers to tests and run only those tests which contain specific marker, and skip the execution of specific tests unconditionally
- debug test scripts using the python debugger
- parametrize calls to a function an application using the pytest parametrize decorator
- use fixtures to define common operations for test functions
- apply module and function level scopes for fixtures functions
- bundle fixture functions which are shared across scripts within a single conftest.py file
- summarize the key concepts covered in this course



Python Unit Testing: Testing Python Code Using doctest

objectives

- write and execute a test using the doctest module
- recognize where doctests can be placed within source code
- create a readme file for a Python module which bundles documentation and testing in a single file
- use the ELLIPSIS directive to account for unpredictable output in doctests
- use the ELLIPSIS directive to check if the code raises exceptions for incorrect input values
- recall how to set doctest to ignore whitespace in the output of tests tests
- summarize the key concepts covered in this course



Python Requests: HTTP Requests with Python



Flask in Python: An Introduction to Web Frameworks & Flask

objectives:

- install the Python Requests package and set up a workspace
- make a GET request and explore the response object returned, which includes a status code and headers
- parse a response body containing JSON-formatted data
- invoke a GET request that includes parameters
- use the Requests package to construct a POST request that includes a set of key-value pairs to be submitted to a server
- submit data to pastebin.com using their APIs and with a POST request
- retrieve metadata for a resource by means of a HEAD request
- invoke PUT, OPTIONS, and DELETE requests and recognize the specific use case for each of them
- set the headers in an HTTP Request and parse the headers in a Response object
- define the encoding format for the contents in an HTTP Response and download and handle binary content such as images
- identify the format of an HTTP Response and parse the data accordingly
- handle successful and unsuccessful HTTP requests according to the status code of the response
- recognize when an HTTP request has triggered a redirect, explore the redirect history, and configure the read and connect timeout values for a request
- catch errors that are thrown by HTTP requests by using the exceptions module in the Requests package
- compare the GET and HEAD requests, summarize the PUT and DELETE requests, and view the redirect history following a GET request

objectives:

- describe the steps involved in a web request and the role of web applications in the process
- recognize the various pieces that can make up a web applications and the role of web frameworks in defining them
- recall the features of the Flask framework that are available either out of the box or via extensions
- describe the roles of routes in a Flask application and the options available when defining a route function
- recognize the need for templates when defining a web site and describe the use of Jinja for this purpose
- identify some of the commonly used extensions in Flask applications and recall the purpose they serve
- summarize the key concepts covered in this course



Flask in Python: Building a Simple Web Site Using Flask



Flask in Python: User Interactions in Flask Applications

objectives:

- install Flask in a virtual environment on your development machine
- write the code for a simple "Hello World" web site using Flask
- recognize how route definitions can be altered and the benefits of running your Flask app in debug mode
- define a route that renders an HTML page when a URL is accessed
- download and use some boilerplate HTML files so that your web site definition need not begin from scratch
- modify the boilerplate CSS and HTML definitions to customize the look of a web site
- generate URLs dynamically using the `url_for` function
- create a base template that can be inherited by other templates, along with placeholders that can be overridden
- inherit the elements from a base template in a child template HTML file
- define multiple routes to point to the same route function
- summarize the key concepts covered in this course

objectives:

- serve a custom error page whenever a 404 error is invoked on your web site
- configure a route in your Flask app so that POST requests can be submitted to it
- use the Flask debugger to record information in your application's log
- convey the invocation of an operation to end users using message flashing
- highlight flashed messages by defining a style for them in a CSS file
- install and use WTForms in your Flask application to accept user input for registration
- define a login page using field definitions and built-in validators available in WTForms
- include the two pages defined using WTForms in the Flask application
- invoke the validators defined for WForm elements to ensure that the user input is in the correct form
- ensure that all the built-in validators applied on the WForm elements work as they are expected to
- summarize the key concepts covered in this course



Flask in Python: User Authentication in a Flask Application



Python Concurrent Programming: Introduction to Concurrent Programming

objectives:

- install SQL Alchemy and use it to connect the Flask application to a SQLite database
- use SQL Alchemy to generate relational database tables for each model defined in your application
- execute queries against tables using a SQLAlchemy model
- separate the model definitions, routes, and app initialization into separate files that are easier to maintain
- modify the import statements to account for the restructuring of the Flask app and test that the functionality has not been affected
- use the Bcrypt package to generate hashes of passwords so that they can be stored securely
- create a bespoke validator for fields in your WTForms
- install and use a Flask extension to allow users to login to your Flask app and maintain a login session
- implement the logout feature so that users can sign out of a Flask app
- access and display the three most recent reviews posted on your Flask web site
- test that the feedback display functionality implemented works as expected
- configure your Flask web site to display images loaded from the app's static resources
- summarize the key concepts covered in this course

objectives:

- recognize what sequential execution is and what its limitations are
- describe multithreading and compare its performance with a sequential execution of tasks
- identify the specific use cases for multithreading
- summarize multiprocessing and contrast it with multithreading
- describe the implementation of threads and processes in the Python language
- recognize what a race condition is and when it can occur with concurrent programming
- outline how locks can help concurrent tasks synchronize their actions on shared resources
- summarize how semaphores can restrict the number of concurrent tasks accessing a shared resource
- identify the use cases for event and condition objects in Python and distinguish between the two
- recognize when a deadlock can occur in an application and the actions you can take to avoid it
- enumerate the built-in data structures available in Python for concurrent programming
- outline how pools of threads and processes can optimize concurrency in your application
- recall the different synchronization mechanisms in Python and the conditions necessary for deadlocks



objectives:

- initialize and execute a thread in Python
- set a name for a thread and ensure a thread waits for a created thread to complete
- define your own type to run in a thread by deriving the `threading.Thread` class
- execute two threads concurrently to save time relative to a sequential execution
- identify what a race condition is and when this may occur with multithreading
- use a lock to synchronize threads that update a shared resource
- recognize the conditions under which a deadlock may occur
- define tasks in such a manner that deadlocks will not occur
- create semaphores in Python and recognize the effect of calling their `acquire` and `release` methods
- describe the use of `BoundedSemaphores` to restrict the number of times a semaphore can be released
- define threads that will wait for an event to occur before they can proceed
- create a condition instance that will have multiple threads waiting for data to be generated
- recall how to retrieve details about running threads and describe semaphores in Python

objectives:

- describe the different types of Queue objects available in Python
- implement producer and consumer threads that add to and remove from a queue
- define, initialize, and execute a process in Python
- distinguish between multiprocessing and multithreading in the context of sharing data in memory
- share data between processes using shared memory
- share complex Python data between processes using the `Manager` class
- implement locks to allow only one process to update a shared resource
- use queues and pipes to enable communication between concurrent processes
- recall the ways in which processes can share data and distinguish between Queue and Pipe

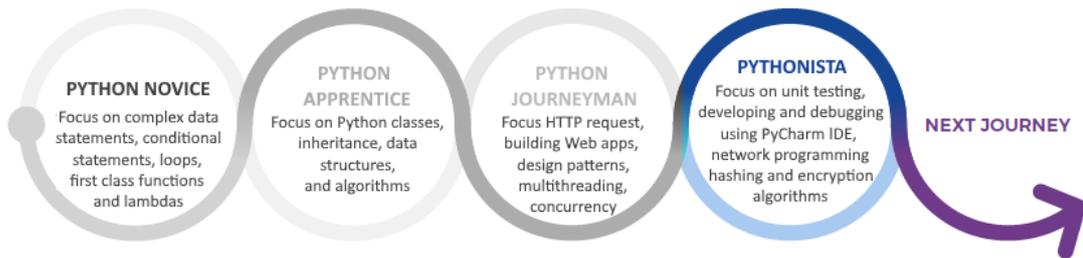


objectives:

- create a pool of processes to which tasks can be submitted
- use the map function of a `multithreading.Pool` instance to submit multiple tasks to a process pool
- recognize the use of futures objects to execute tasks asynchronously
- compare the performance of process pools and thread pools for tasks that are network-bound
- compare the performance of process pools and thread pools for tasks which are CPU-bound
- create and execute a coroutine using the `asyncio` module
- use the `run`, `create_task`, and `gather` functions in the `asyncio` modules to execute tasks
- summarize process pools and contrast multithreading and multiprocessing in Python

Track 4: Pythonista (Duration: 6h 13m 36s)

PYTHON NOVICE TO PYTHONISTA



Excel with Python: Working with Excel Spreadsheets from Python



Excel with Python: Performing Advanced Operations

objectives:

- create a Microsoft Excel workbook by choosing from a list of templates
- create a Microsoft Excel workbook using the openpyxl library
- access individual cells programmatically using openpyxl
- access entire rows and columns and manipulate them programmatically
- save the contents of an in-memory representation to a Microsoft Excel file on disk
- use lists and other Python containers to read to and write from Excel files
- specify rows and columns that always ought to be on-screen as a user browses an Excel file
- filter data in a range based on various attributes
- sort data in a range using different columns and sort criteria
- customize the size of rows and columns to enhance readability
- merge and unmerge groups of cells
- summarize the key concepts covered in this course

objectives:

- apply styling elements to control the display of data in cells
- apply sophisticated styles and alignments to format cell contents
- use number formats to represent currencies and add comma separators
- apply formatting that varies based on the value contained in a cell
- choose from different in-built icon sets and rules to control cell formatting at a granular level
- insert images into Microsoft Excel files and control their size and location
- insert formulae into Excel workbooks
- use openpyxl to programmatically construct formulae in workbooks
- use the \$ operator to convert relative cell references into absolute ones
- use openpyxl to construct both absolute and relative cell references
- use VLOOKUP to lookup specific values from a range in Excel
- assign names to groups of cells and use those names in formulae to enhance readability
- use Excel pivot tables to dynamically analyze and group data
- use Pandas to read data from Microsoft Excel and perform pivoting operations
- use Pandas to perform multi-level indexing and access individual row values as well as index values
- summarize the key concepts in this course



Excel with Python: Constructing Data Visualizations

objectives:

- use Microsoft Excel to visualize data
- use openpyxl to construct visualizations in Excel workbooks
- tweak the title, formatting, and legend of a graph
- alter the data displayed on the axes of a graph
- alter the weight and line style of data series to customize the appearance of a visualization
- construct and modify bar chart visualizations in Excel
- use openpyxl to build different types of bar charts programmatically
- plot financial data containing open, high, low, close, and volume information about stock prices
- use bubble charts to represent three dimensions of data in a two dimensional visualization
- summarize the key concepts in this course



Socket Programming in Python: Introduction

objectives:

- use the socket module in a Python application and identify the functions that can be used to get information about the application's host
- write server and client applications that can communicate with each other using TCP sockets
- implement socket communication between applications by using Python's with context manager
- configure an application to wait for a set amount of time for communication from another process
- recognize how data needs to be transformed to bytes when transferring it over Python sockets
- use the pickle module to serialize data when sending it over a socket connection and de-serialize the data at the other end
- recover and use objects that were transmitted from another Python application over a socket connection
- summarize the key concepts covered in this course



Socket Programming in Python: Advanced Topics

objectives:

- build a Python app to break up a large text file into chunks and send the chunks over a socket connection to a recipient app
- code a Python app to receive a large text file in chunks and reconstruct that file
- transmit an image file from one Python app to another by breaking it up into chunks
- configure the server of a client-server chat application
- write the code for the client end of a client-server chat application
- recognize the effects of setting sockets to run in blocking mode when large transfers are involved
- recall the considerations for setting a Python socket to use non-blocking mode
- write a Python app that subscribes to RSS data feeds
- set up applications to transfer data using UDP and distinguish between UDP and TCP sockets
- summarize the key concepts covered in this course

Business Skills for Pythonista

Business & Leadership for Pythonista (Optional)

 <p>COURSE</p> <p>Developing and Supporting an Agile Mind-set</p> <p>177</p>	 <p>COURSE</p> <p>Encouraging Team Communication and...</p> <p>75</p>	 <p>COURSE</p> <p>The Essential Role of the Agile Product Owner</p> <p>51</p>	 <p>COURSE</p> <p>Using Strategic Thinking to Consider the Big Picture</p> <p>80</p>	 <p>COURSE</p> <p>Getting to the Root of a Problem</p> <p>138</p>
 <p>COURSE</p> <p>Unleashing Personal and Team Creativity</p> <p>87</p>	 <p>COURSE</p> <p>Contributing as a Virtual Team Member</p> <p>43</p>	 <p>COURSE</p> <p>Developing a Growth Mind-set</p> <p>206</p>	 <p>COURSE</p> <p>Developing a Successful Team</p> <p>39</p>	 <p>COURSE</p> <p>Reaching Sound Conclusions</p> <p>16</p>

Productivity Tools for Pythonista

Productivity Tools for Pythonista (Optional)

 <p>COURSE</p> <p>Signing in & Navigating within Spaces</p> <p>4</p>	 <p>COURSE</p> <p>Setting Up & Managing Spaces</p> <p>2</p>	 <p>COURSE</p> <p>Working with Spaces</p> <p>2</p>	 <p>COURSE</p> <p>Working with Team Members</p> <p>14</p>	 <p>COURSE</p> <p>Configuring Spaces</p> <p>1</p>
 <p>COURSE</p> <p>Signing in & Setting Up</p> <p>2</p>	 <p>COURSE</p> <p>Using Channels</p> <p>2</p>	 <p>COURSE</p> <p>Private Messaging & Communication Tools</p> <p>6</p>	 <p>COURSE</p> <p>Creating, Finding & Sharing Information</p> <p>1</p>	 <p>COURSE</p> <p>Configuring Slack</p> <p>9</p>
 <p>COURSE</p> <p>Using the iOS App</p> <p>1</p>				

Books

Bookshelf ⓘ Optional

- The Quick Python Book, Third Edition** (4 likes)
- Beginning Python: From Novice to Professional, Thir...** (29 likes)
- Python Basics** (10 likes)
- Beginning Programming with Python for Dummies, 2nd...** (6 likes)
- Python Programming: An Introduction To Computer...** (2 likes)
- Python Crash Course: A Hands-on, Project-Based...** (82 likes)
- Practical Python Design Patterns: Pythonic Solution...** (1 like)
- Asynio Recipes: A Problem-Solution Approach** (1 like)
- Pro Python Best Practices: Debugging, Testing and...** (3 likes)
- Cracking Codes with Python: An Introduction to Building...** (1 like)
- Python Unit Test Automation: Practical...** (1 like)

Follow us on:



SkillTech
Technology hired for excellence

www.skilltech.pl

email: biuro@skilltech.pl

tel. 22 44 88 827