codetoday

Curriculum

Coding in Python

Codetoday Curriculum

Introduction

At codetoday we have developed an extensive curriculum that ranges from the very basics and fundamental principles all the way to intermediate and advanced areas of programming across a broad range of programming applications.

Our focus is on teaching programming thoroughly. We have designed a path through the topics available in coding that enables us to rapidly move to more complex and engaging projects. Our approach is to introduce the fundamentals very early on in their most basic form, and then to revisit these topics adding breadth and depth later on as students become more confident and proficient.

An important aspect throughout our whole curriculum is the focus on best practices and neat and efficient coding styles. Often beginners write inefficient code as this has little or no effect on simple, short programs. We feel that this should be corrected early on, before it becomes a problem (when programs become longer and more complex). We therefore model and discuss best coding practices right from the very beginning and throughout all our courses.

Curriculum

In this document we outline the curriculum we use in all codetoday courses. *The Beginner and Intermediate Stages are shown here*, as well as an indication of some of the topics we cover at Advanced stages (in the Further Topics section).

Our approach is to find the right balance between introducing new topics and consolidating existing ones. A common misconception that students can have is that if they have used a certain tool in the past, they mistakenly think that they *know* that topic in coding. In reality each topic always has more depth and more complex uses that need to be learnt once the fundamentals are well understood. We therefore return to topics in later Stages to dive deeper into them.

An important aspect when moving from beginner to intermediate is the ability to deal with more complex projects. The topics in the early Stages may be well understood by themselves, but combining them into a complex project requires more expertise.

Progressing through the Curriculum

Our approach is to start from the fundamentals in Stages 1-3 for all age groups. This allows us to ensure that these key topics are understood well and thoroughly as everything else relies on them later on. However, we progress at different paces for different age groups. Older students can go through Stages 1-4 very rapidly before we slow down to spend more time on the more complex topics. With younger students we move more slowly as they need more time to master the basics.

Stage 1: General concepts and `for` loop (basic)

Key topics: code structure - `for` loop - commenting

Introduction to coding/programming and to programming languages
Concept of built-in commands / importing modules
Basic Python syntax
Basic structure and logical order of code
DRY: Don't Repeat Yourself
`for` loop — basic: using `range()`
`for` loop: syntax
`for` loop: identifying when to use it
Further Python syntax: colon and indent
Commenting, keeping code well structured, naming good practices

LEARNING OBJECTIVES

Students will be able to:

- Write basic Python syntax correctly
- Place lines of code in the correct order for simple programs
- Translate simple instructions into Python code
- Identify when to use a `for` loop
- Write a basic `for` loop (using `range()`)
- Understand the purpose of the indent in Python

Stage 2: Variables, `while` loop (basic) and defining functions using `def` (basic)

Key topics: variables - `while True` loops - `def` for basic function definitions

Assignment of data / variables
`while` loop — basic: `while True`
Defining basic functions using `def` (with no input parameters and no return values)
Understanding some of the most basic/common error messages
`print()`

LEARNING OBJECTIVES

Students will be able to:

- Have a good understanding of the structure of a computer program
- Identify when simple data needs to be stored using a variable
- Write comments in their code without being prompted to do so
- Identify when to use a `while True` loop
- Write a `while True` loop
- Notice errors and identify their location within code

Stage 3: Data types, if statements and conditional `while` loops

Key topics: `if` statements - loops with conditions

Understanding basic error messages and how to fix them
Revisiting storing data using variables (=)
Introduction to basic data types (`int`, `float`, `str`, `bool`)
`if` statements
Equality and comparison operators (==, <, >)
Introduce text-based programs
while loop with conditional statements and/or Boolean flags
`input()`
Introduce The White Room analogy
String formatting

LEARNING OBJECTIVES

Students will be able to:

- Understand the purpose of defining functions and write a basic function definition
- Identify when to use an `if` statement
- Write an `if` statement including an equality or comparison operator
- Write a `while` loop with a conditional statement and/or Boolean flag
- Create variables when required, using good naming practices
- Identify and fix some basic errors independently

Stage 4: Lists and iterating through lists with `for` loops

Key topics: `lists` and data types

Review basic data types
Revisit decomposing ideas and processes into individual, unambiguous sequential steps
`lists`*: basic introduction
`lists`: accessing data
`lists`: introduce basic `list` methods
`for` loop: iterating through a `list`
Using `lists` and `for` loops
Focus on understanding errors
** Note: lists can be used during earlier stages but without going into any details. This stage is when we start to fully introduce lists*

LEARNING OBJECTIVES

Students will be able to:

- Understand the need for lists and create a list using `[]`
- Identify when to iterate using `range` and when directly through a list when using `for` loops
- Understand indexing, including zero-indexing
- Identify and name different data types
- Have a basic understanding of initialising an empty list and then populating it using a `for` loop
- Write down clear steps when planning a project

Stage 5: Functions with parameters and further list consolidation, debugging

Key topics: `lists` *and functions - errors, bugs and debugging*

Introduce the SRDR concept (**S**tore | **R**epeat | **D**ecide | **R**euse)
Review and extend using lists to collect like items in a single data structure
Using the `time` module for timing actions
Consolidate the use of Boolean flags to control execution of subsections in code
Defining functions with parameters and input arguments*
Basic introduction to scope and namespace
Further work on understanding and dealing with errors
Introduce concept of bugs
Debugging using the print() statement
** Note: For some groups this can be introduced in conjunction with return statements and other related topics in Stage 6*

LEARNING OBJECTIVES

Students will be able to:

- Identify when to use a list
- Access data from lists through indexing, including using the -1 index
- Write a for loop that iterates through a list
- Manipulate lists by using `append()` and `remove()`
- Initialise an empty list and populate it by using a `for` loop
- Recognise function definitions that have parameters and modify such function definitions

Stage 6: Functions: return statements and creating instance attributes*

Key topics: `lists` and functions

Self-contained nature of functions and the local nature of variables within function definitions
Review parameters in function definitions, including default values
Returning data from functions using a `return` statement
The White Room Analogy
Attaching data to an existing object using the dot notation *(creating instance attributes)**
Data types: distinction between mutable and immutable data types
tuples**
Further work on understanding and dealing with errors
** Knowledge of Object-Oriented Programming (OOP) not required at this stage, however we have been using objects since Stage 1 when creating a turtle.Turtle. Attaching variables to an object using the dot notation can be introduced here without the need for OOP*
** * Can be introduced when introducing dictionaries in Stage 7*

LEARNING OBJECTIVES

Students will be able to:

- Understand that variables created inside function definitions are local, and what this means
- Use a return statement in function definitions and call functions which have `return` statements, understanding how data is transferred
- Identify what data type or data structure is required for simple data storage requirements (excluding nested data structures)
- Have a general understanding of mutable and immutable data types and how this affects how we use their respective methods
- Create instance attributes (nomenclature not required) to move data between functions in programs already containing objects (e.g. turtle.Turtle / turtle.Screen)

Stage 7: Dictionaries and nested data structures

Key topics: dictionaries

Reinforce commenting and structuring code neatly
Focus on translating ideas into Python code
dictionaries: creating and accessing data
dictionaries: adding data to existing dictionaries through assignment
dictionaries: basic methods for dictionaries
list of dictionaries and the concept of nested data structures
Debugging using Visual/IDE debuggers

LEARNING OBJECTIVES

Students will be able to:

- Independently add appropriate comments and structure code efficiently
- Understand the structure of dictionaries and create a dictionary using {}
- Identify when to use a dictionary
- Manipulate data in a dictionary by adding, removing and changing items in the dictionary

Stage 8: Reading and writing to file (.txt and .csv), introduction to data analysis

Key topics: data manipulation and data structures

Data types: a very good understanding of distinct types
Review dictionaries
`for` loop — looping through a dictionary
Algorithm building and planning
Reading from text files (.txt)
Converting data from one form to another
Concept of cleaning data before using
Focus on text-based programs and programs to analyse data
Reading from spreadsheets (.csv)
Writing data to file (.txt and .csv)
docstrings: documenting functions
`list` comprehensions

LEARNING OBJECTIVES

Students will be able to:

- Have a good general awareness of all variables in their programs and their corresponding data types
- Independently plan algorithms of intermediate complexity
- Open a file and read data from it into their Python program
- Open a writable file and write data from their program into the file
- Understand how to move data from one form (data type) into another
- Understand the overall need to clean data obtained from any source before using it

Stage 9: Basics of Object-Oriented Programming

Key topics: OOP

Data and manipulation of data
Philosophy of OOP
Creating classes
Interaction between different objects
Magic (dunder) functions
Inheritance

LEARNING OBJECTIVES

Students will be able to:

- Understand what OOP is and when it may be useful
- Create a simple class with `__init__()` method and additional methods
- Identify inheritance of classes
- Identify when magic functions are used

Stage 10: Numerical programming, visualisation, vectorisation of equations

Key topics: numpy and matplotlib

Introduce `numpy`
Introduce `matplotlib`
import formats

LEARNING OBJECTIVES

Students will be able to:

- Have a basic understanding of `numpy` and `matplotlib`, and the `ndarray` data type
- Plot simple graphs

Further Topics

- Reading data from APIs
- Further work on OOP
- Further work on `numpy` and `matplotlib`
- Functional Programming
    - philosophy and rules for FP
    - `map()` and `filter()`
    - `lambda` functions
- Generators
- Efficiency of different coding styles
- `*args` and `**kwargs`
- `pandas` and complex data manipulation and analysis
- Scientific/mathematical modelling

codetoday

Codetoday Limited

13 Hawley Crescent, London,
NW1 8NP, United Kingdom

T 020 3289 7431
E info@codetoday.co.uk
W codetoday.co.uk

Codetoday Limited is a company registered in England (company number 9789836).
Registered office: 13 Hawley Crescent, London, NW1 8NP, United Kingdom