Course Syllabus

PCAP | Programming Essentials in Python

**0. Introduction**

0.1. Programming – absolute basics

0.1.1. How does a computer program work?

0.1.2. Natural languages vs. programming languages

0.1.3. Compilations vs. interpretation

0.2. Python – a tool, not a reptile

0.2.1. What is Python?

0.2.2. Who is Python's creator?

0.2.4. Why Python?

0.2.5. Why not Python?

0.3. There is more than one Python

0.3.1. Python 2 vs. Python 3

0.3.2. Python aka Cpython

0.3.3. Cython

0.3.4. Jython

0.3.5. PyPy and Rpython

0.4. Begin your Python journey

0.4.1. How to get Python and how to get to use it

0.4.4. How to write and run your very first Python program

0.4.8. How to spoil and fix your code

0.5. Edube Sandbox and Labs

**PART 1: BASICS**

**1. Basics I**

1.1 Your first program

1.1.1 Your first program

1.1.2 The print() function – how the computer talks to you

1.1.9 The print() function – formatting the output

1.2 Python literals

1.2.1 Literals – what they are anyway?

1.2.2 Literals – integers

1.2.8 Literals – floats

1.2.15 Literals – strings

1.2.22 Literals – Boolean values

1.3 Operators – data manipulation tools

1.3.1 Operators and expressions

1.3.2 Arithmetic operators

1.3.24 Operators and their priorities

1.3.25 Operators and their bindings

1.4 Variables – data-shaped boxes

1.4.1 Variables – how to name them

1.4.2 Variable names vs. Python keywords

1.4.3 How to assign a variable

1.4.10 How to comment your code

1.4.11 Shortcut operators

1.5 How to talk to computer?

1.5.1 Output vs. input

1.5.2 How to input data with the input() function

1.5.6 How to convert strings into numbers

1.5.7 Some simple interactive programs

1.5.7 String operators

1.5.16 How to convert numbers into strings

**2 Basics II**

2.1 Making decisions in Python

2.1.1 How to ask questions and how to get answers

2.1.2 Relational operators

2.1.11 Making use of the answers

2.1.13 Conditions and conditional execution – the if statements

2.1.16 How indentation makes the code

2.1.17 The more conditional execution – if-else statements

2.1.21 The elif clause

2.1.22 Some simple examples

2.2 Python's loops

2.2.1 Looping your code with while

2.2.12 Looping your code with for

2.2.24 Controlling your loops with break and continue

2.3 Logic and bit operations in Python

2.3.1 Computer logic and its operators

2.3.8 Logical values vs. single bits

2.3.9 Bitwise operators

2.3.15 How to deal with single bits

2.4 Lists – collections of data

2.4.1 Lists – why do we need them so much?

2.4.2 How to create a list

2.4.3 How to use a list

2.4.8 Removing elements from a list

2.4.9 How not to use a list

2.4.12 List methods – methods vs. functions

2.4.14 Adding elements to lists

2.4.22 Making use of lists

2.4.23 The second face of the for loop

2.3.24 Lists in action

2.5 Sorting simple lists – the bubble sort algorithm

2.6 Lists – some more details

2.6.1. How lists are stored

2.6.2. Slices – the powerful tools

2.6.14. The in and not in operators

2.7 Lists in advanced applications

2.7.1 Lists in lists

2.7.2 The list comprehension: why and how

2.7.3 Lists in lists – matrices

2.7.10 Lists in lists – 3rd dimension

**3 Basics III**

3.1 Writing functions in Python

3.1.1 Functions: why do we need them?

3.1.5 Where do functions come from?

3.1.6 Your first function

3.2 How functions communicate with their environment

3.2.1 Parametrized functions

3.2.2 How to define and use function parameters

3.2.6 What is shadowing?

3.2.7 Positional arguments

3.2.12 Keyword arguments

3.2.14 Mixed arguments

3.2.19 Setting parameters' default values

3.3 Returning a result from a function

3.3.1 A function's effects and results – the return statement

3.3.4 Returning a value

3.3.8 The None value

3.3.9 Returning the non-None value

3.3.11 Argument vs. parameter compatibility

3.3.12 A list as a function's result

3.4 Scopes in Python

3.4.1 Functions and scopes

3.4.2 How do scopes work?

3.4.5 How to make a variable global

3.4.6 How the parameters interact with their arguments

3.5 Creating functions

3.5.1 Some exercises with designing and writing functions

3.5.22 Recursion – how to make a function more powerful?

3.6 Tuples and dictionaries

3.6.1 Sequence types and mutability

3.6.2 What is a tuple?

3.6.3 How to create a tuple

3.6.5 How to use a tuple

3.6.11 What is a dictionary?

3.6.12 How to make a dictionary

3.6.13 How to use a dictionary

3.6.21 How a dictionary and a tuple can work together

**PART 2: INTERMEDIATE**

**4. Intermediate I**

4.1 Using modules

4.1 Using modules

4.1.1 What is a module?

4.1.2 How to make use of a module?

4.1.4 Importing a module

4.2. Some useful modules

4.2.1 Working with standard modules

4.2.4 Some functions from the math module

4.2.9 Some functions from the random module

4.2.15 Some functions from the platform module

4.3 What is a package?

4.3.1 Modules and packages

4.3.2 Your first module

4.3.14 Your first package

4.4 Errors – a programmer's daily bread

4.4.1 Errors, failures, and other plagues

4.4.2 Exceptions

4.5 The anatomy of an exception

4.6 Some of the most useful exceptions

4.7 Characters and strings vs. computers

4.8 The nature of Python's strings

4.9 String methods

4.10 Strings in action

4.10.1 Comparing strings

4.10.7 Sorting strings, and not only strings

4.10.10 Strings vs. numbers

4.11 Four simple programs

4.11.1 Caesar’s cipher – the coder

4.11.2 Caesar’s cipher – the decoder

4.11.3 Extracting numbers from a line of text

4.11.4 Checking the IBAN

**5. Intermediate II**

5.1 Basic concepts of object programming

5.1.1 What is an object?

5.1.5 The object – what is it again?

5.1.7 What does an object have?

5.1.8 Your first class

5.2 A short journey from the procedural to the object approach

5.2.1 What is a stack?

5.2.2 The stack – a procedural approach

5.2.7 The stack from scratch

5.3 Properties

5.3.1 Properties in detail

5.3.2 Instance variables

5.3.4 Class variables

5.3.8 Checking an attribute's existence

5.4 Methods

5.4.1 Methods in detail

5.4.a The inner life of classes and objects

5.4.b Reflection and introspection – two names of the same phenomenon

5.4.c Investigating classes – what can we find out about them?

5.5 Inheritance – one of object programming foundations

5.5.1 Inheritance – why and how

5.5.a How Python finds properties and methods

5.5.b How to build a hierarchy of classes

5.5.c Inheritance vs. composition

5.5.d Single inheritance vs. multiple inheritance

5.5.e Diamonds and why you don't want them

5.6 Exceptions once again

5.6.1 Exceptions are classes

5.6.a Detailed anatomy of an exception

5.6.b How to create your own exception

5.6.c How to use your own exception

5.7 Generators and closures

5.7.1 Generators – where to find them

5.7.a The yield statement

5.7.b How to build your own generator

5.7.c More about list comprehensions

5.7.d The lambda function

5.7.e How and when to use lambdas

5.8 Processing files

5.8.1 Accessing files from Python code

5.8.2 File names

5.8.6 File streams

5.8.9 File handles

5.8.13 Opening the streams

5.8.18 Selecting text and binary modes

5.8.19 Opening the stream for the first time

5.8.20 Pre-opened streams

5.8.24 Closing streams

5.8.26 Diagnosing stream problems

5.9 Working with real files

5.9.1 Dealing with text files

5.9.6 How to work with binary files

5.9.7 How to read bytes from the stream

5.9.a How to write bytes from the stream

5.9.b Copying files – a simple, but functional tool