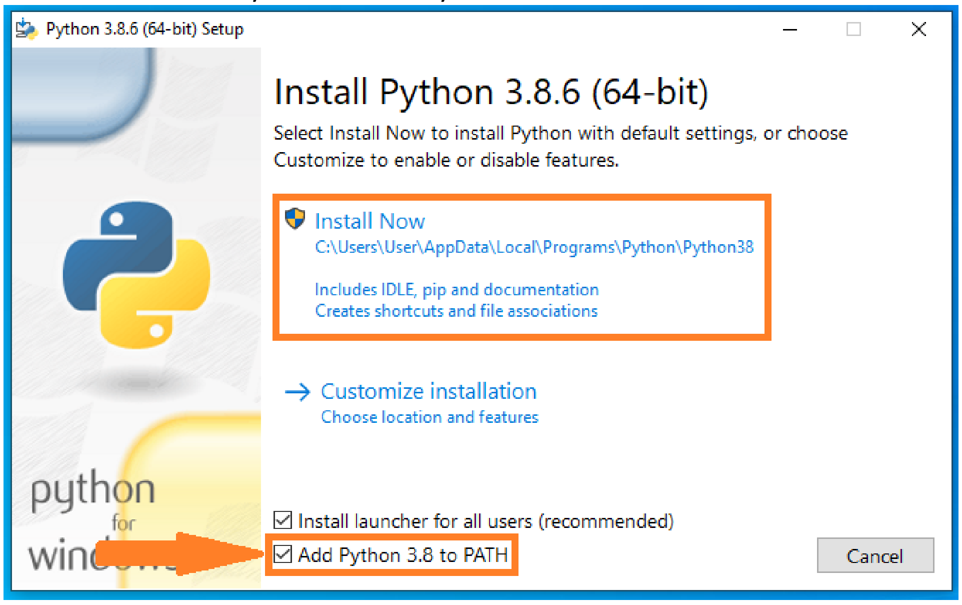
Guide to Grande Omega (GO)

Prerequisites for Dev 1 and Dev 2:

The installation procedure is slightly different for Windows and Mac users, so please follow the one that is for your Operating System.

1) Windows users:

- Download and install **node.js** from: https://nodejs.org/en/download/
- Download and install **Python 3.8** from: https://www.python.org/downloads/
    - Make sure to have Python added to your environment variables



Python 3.8.6 (64-bit) Setup

Install Python 3.8.6 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\User\AppData\Local\Programs\Python\Python38

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.8 to PATH

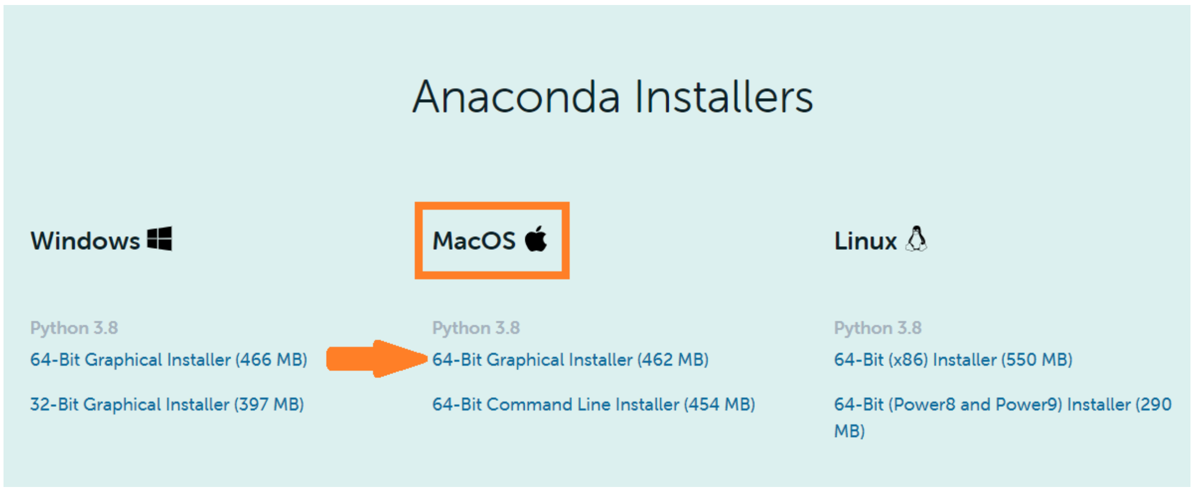
Cancel

NOTE : clicking the box as shown in the previous image this will work most of the times. You can check if Python was successfully added to the environment variables by following the instructions in the following link: https://www.architectryan.com/2018/03/17/add-to-the-path-on-windows-10/)

2) Mac users:

- Download and install **node.js** from: https://nodejs.org/en/download/
- Download and install **Anaconda** from: https://www.anaconda.com/products/individual/

By clicking on this link, you will be directed to the Anaconda download website and there you need to find the Anaconda installer for MacOS and download it. There should be two installers and for having an easy installation pick Graphical Installer.

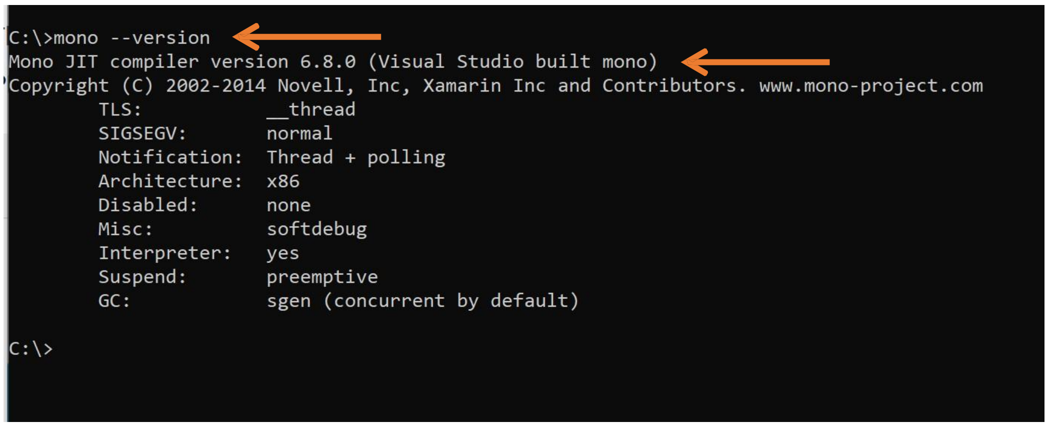


Anaconda Installers

Windows	MacOS	Linux
Python 3.8	Python 3.8	Python 3.8
64-Bit Graphical Installer (466 MB)	64-Bit Graphical Installer (462 MB)	64-Bit (x86) Installer (550 MB)
32-Bit Graphical Installer (397 MB)	64-Bit Command Line Installer (454 MB)	64-Bit (Power8 and Power9) Installer (290 MB)

Prerequisites for Dev 3 and Dev 4:

- Download and install **node.js** from: https://nodejs.org/en/download/
- Download and install **Mono** from: https://www.mono-project.com/download
  - ✓ See Osiris for the version you should use
  - ✓ Important, download the 32bit version, **NOT** the 64bit
  - ✓ For both Windows and Mac users make sure to have Mono installed and set up properly. Run *mono --version* on the console to verify the correct installation and version.



```
C:\>mono --version
Mono JIT compiler version 6.8.0 (Visual Studio built mono)
Copyright (C) 2002-2014 Novell, Inc, Xamarin Inc and Contributors. www.mono-project.com
        TLS:           __thread
        SIGSEGV:       normal
        Notification:  Thread + polling
        Architecture:  x86
        Disabled:      none
        Misc:          softdebug
        Interpreter:   yes
        Suspend:       preemptive
        GC:            sgen (concurrent by default)

C:\>
```

This is the output you get when calling *mono --version* on Windows. In case Mono is installed, but it does not appear on the console when calling the command above, follow the links below to add the path manually or reinstall Mono:

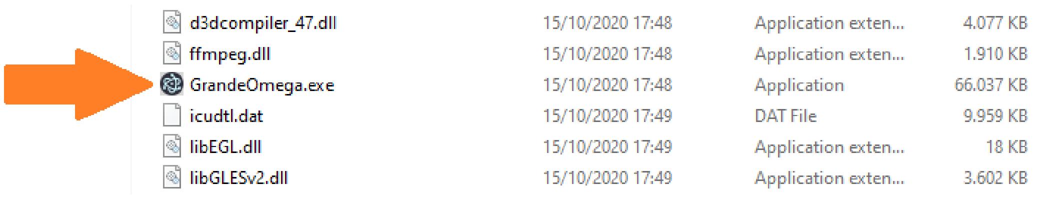
- For Windows users follow this link https://www.architectryan.com/2018/03/17/add-to-the-path-on-windows-10/ to add Mono to the system path. By default Mono is installed in *C:\Program Files (x86)\Mono\bin*,
- For Mac users follow this link https://stackoverflow.com/questions/32542535/how-to-install-mono-on-macos-so-mono-works-in-terminal to add Mono to the system path.

GO Installation:

- Download the client of **GO** from:
  - http://grandeomega.com/downloads/go_student_win.zip (windows)
  - http://grandeomega.com/downloads/go_student_mac.zip (mac)
- **Unzip** the compressed folder downloaded at the previous step
  - Ensure that the location (preferably on the desktop) where you unzip the application to has no spaces in the path name so C:\Users\User1\Desktop\Grande_Omega\go_student_win_tmp2 is *OK*, but C:\Users\User1\Desktop\Grande Omega\go_student_win_tmp2 is *not OK* as there is a space

1) **Windows Users:**

- Execute the **GrandeOmega.exe** file:

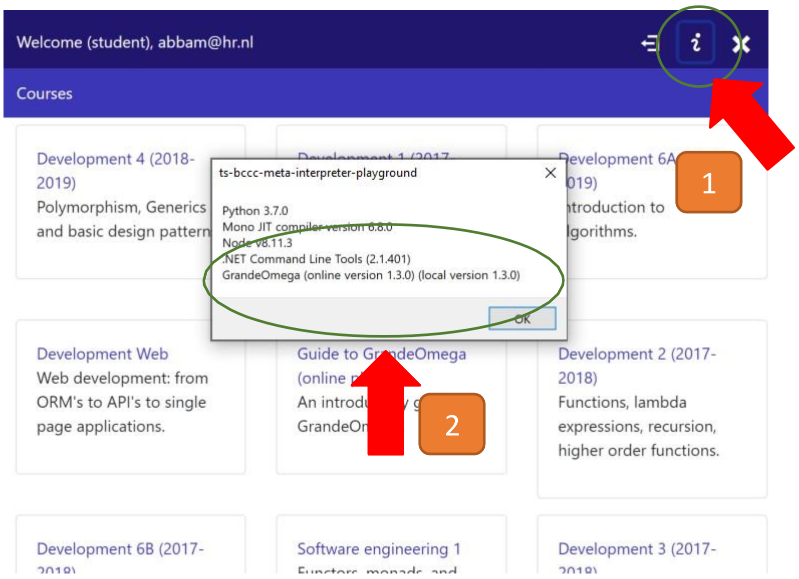


d3dcompiler_47.dll	15/10/2020 17:48	Application exten...	4.077 KB
ffmpeg.dll	15/10/2020 17:48	Application exten...	1.910 KB
GrandeOmega.exe	15/10/2020 17:48	Application	66.037 KB
icudtl.dat	15/10/2020 17:49	DAT File	9.959 KB
libEGL.dll	15/10/2020 17:49	Application exten...	18 KB
libGLESv2.dll	15/10/2020 17:49	Application exten...	3.602 KB

2) Mac users:

- Execute the **start.command** file.
- For Mac users, running *start.command* the first time on your computer will not work due to a security check. To pass this you will need to go to the *Security and Privacy* tab and on the bottom of that window you will find the *start.command* listed with an option next to it: *run anyways*. Click this button to grant GO the permission to run on your computer. Sometime this button is disabled, in this case click the *lock* icon at the end of the same page and then the *run anyways*.

✓ No matter what is your operating system, you can check if everything is correctly set up by clicking the ***" i "*** button on Grande Omega (see following picture):



Development 4 (2018-2019)
Polymorphism, Generics and basic design pattern

ts-bccc-meta-interpreter-playground

```
Python 3.7.0
Mono JIT compiler version 6.8.0
Node v8.11.3
.NET Command Line Tools (2.1.401)
GrandeOmega (online version 1.3.0) (local version 1.3.0)
```

OK

evelopment 6A
019)
ntroduction to
lgorithms.

Development Web
Web development: from ORM's to API's to single page applications.

Guide to GrandeOmega
(online
An introdu
GrandeOm

Development 2 (2017-2018)
Functions, lambda expressions, recursion, higher order functions.

Development 6B (2017-

Software engineering 1

Development 3 (2017-

Troubles with running the assignments?

Windows users:

Try the following steps:

- Ensure you have Mono 6.8, the **32 bit version** (*this is for in case you can't run C# exercises*)
- Ensure GrandeOmega is not in a virtual drive (*for example OneDrive or Google drive*)
- Ensure that the path location of Grande Omega has no spaces
    - Try putting it on the Desktop or on in "C:/"
    - Check for spaces in your username
- Ensure that Mono is available in the command line (you may need to update your path if it is not available (*follow again the prerequisites section at the beginning of this document*)

MacOS users:

Try the following steps:

- Ensure GrandeOmega is not in a virtual drive (*for example OneDrive or Google drive*)
- Ensure **start.command** is executable either by running the command: "sudo chmod a+x start.command"on the terminal in the Grande Omega directory, or by following the last point of section "GO Installation" (for Mac users).

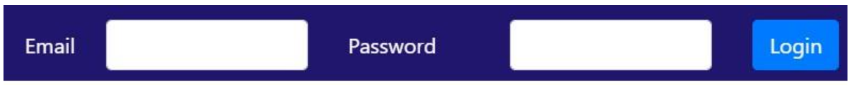
Issue related to Python on MacOS:

For Dev 1 and Dev 2 that you need Python, in some cases, GO runs correctly but it cannot recognize Python. MacOS comes with Python 2 preinstalled but we need Python 3 for the courses. We are not recommending you change the default Python from version 2 to 3 since some of the functionalities of MacOS are depended on Python 2. The best solution is having a Python 3 environment on your system. If you install Anaconda from the beginning, you won't have this problem and you won't need this step, but it can occur if you install standalone version of Python 3. For solving this issue, you need to follow steps explained in the following link. https://opensource.com/article/19/5/python-3-default-mac#what-to-do

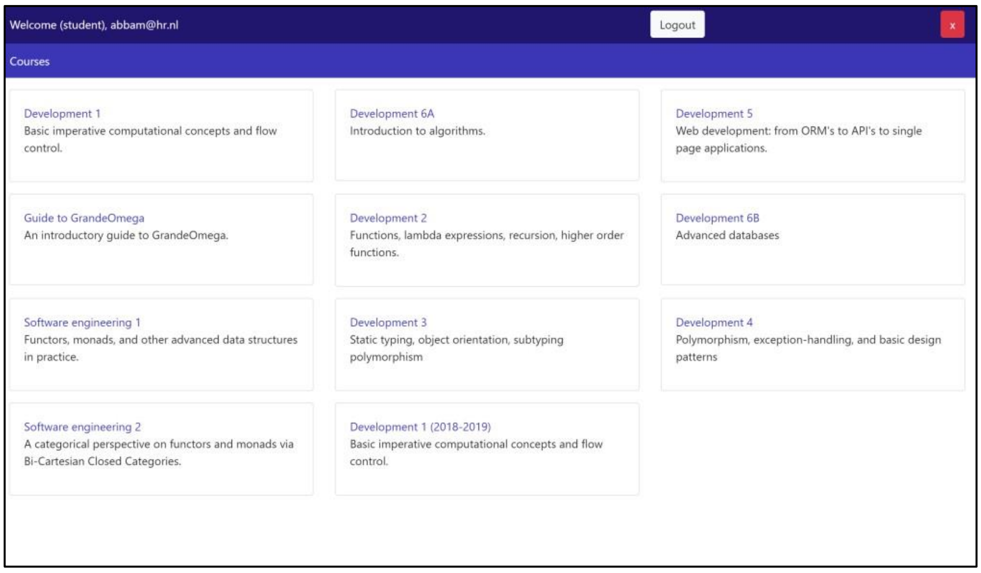
For this solution, you should have "***Homebrew***" already installed. You can find the instruction of the installation in this link: https://osxdaily.com/2018/03/07/how-install-homebrew-mac-os/

Usage

- After the client starts, you need to login with your credentials (you have received via your student email instructions to get access):



- After having logged in, you will see a screen with the courses you are subscribed to:



Development 1
Basic imperative computational concepts and flow control.

Development 6A
Introduction to algorithms.

Development 5
Web development: from ORM's to API's to single page applications.

Guide to GrandeOmega
An introductory guide to GrandeOmega.

Development 2
Functions, lambda expressions, recursion, higher order functions.

Development 6B
Advanced databases

Software engineering 1
Functors, monads, and other advanced data structures in practice.

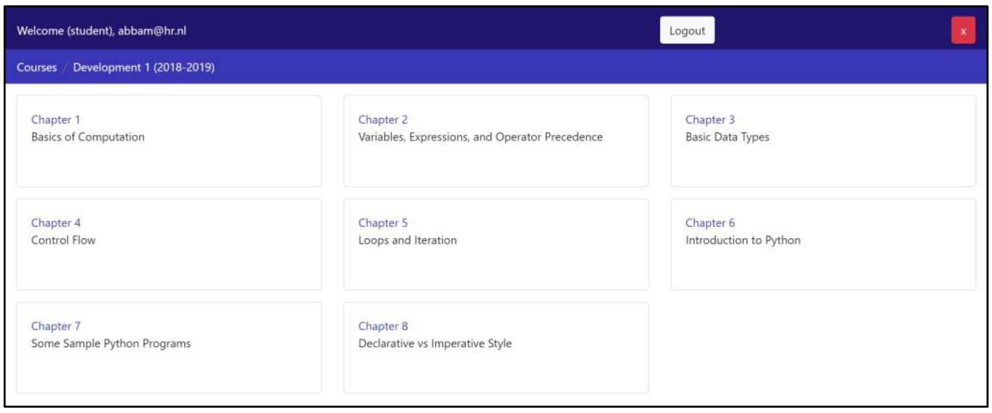
Development 3
Static typing, object orientation, subtyping polymorphism

Development 4
Polymorphism, exception-handling, and basic design patterns

Software engineering 2
A categorical perspective on functors and monads via Bi-Cartesian Closed Categories.

Development 1 (2018-2019)
Basic imperative computational concepts and flow control.

- Clicking on a course, you will see the chapters of materials available for such course:


Welcome (student), abbam@hr.nl
Logout
x
Courses / Development 1 (2018-2019)

Chapter 1
Basics of Computation

Chapter 2
Variables, Expressions, and Operator Precedence

Chapter 3
Basic Data Types

Chapter 4
Control Flow

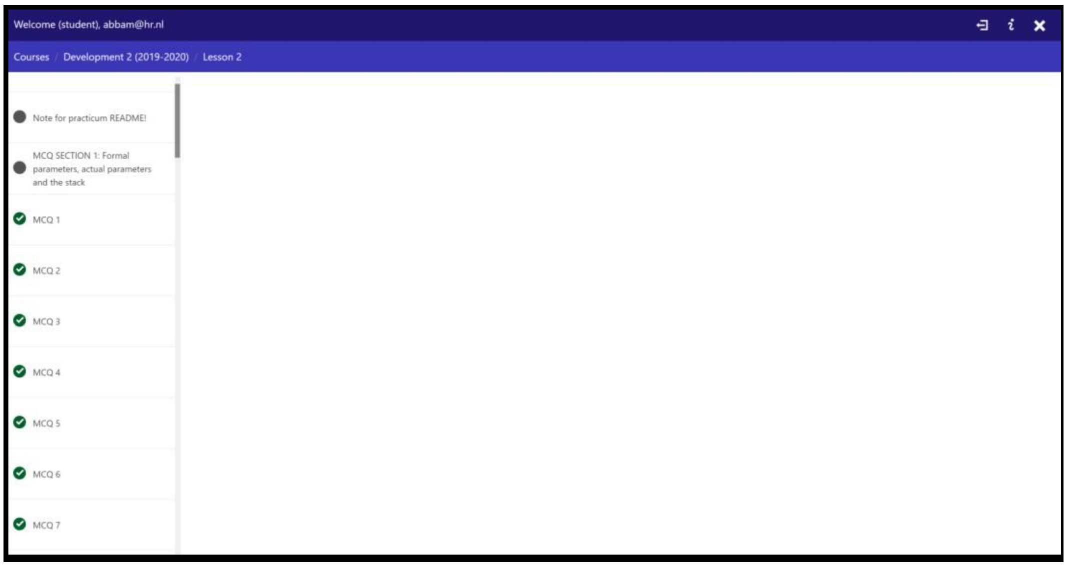
Chapter 5
Loops and Iteration

Chapter 6
Introduction to Python

Chapter 7
Some Sample Python Programs

Chapter 8
Declarative vs Imperative Style

- Clicking on a chapter, you will see the materials associated to such chapter in a column on the left of the screen. Click on the name of an item to open its associated content.



Welcome (student), abbam@hr.nl

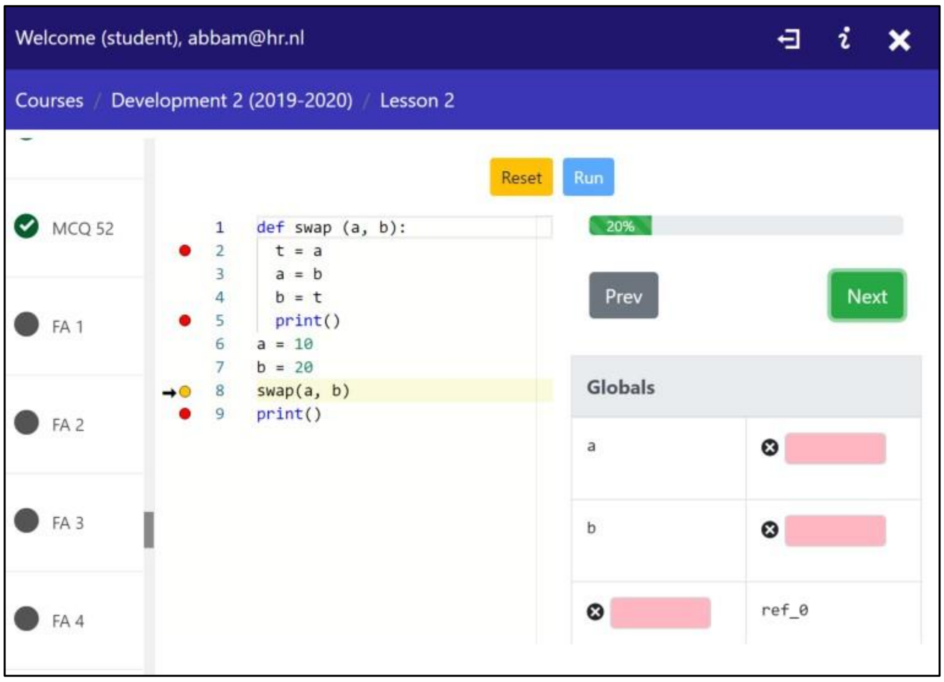
Courses / Development 2 (2019-2020) / Lesson 2

- Note for practicum README!
- MCQ SECTION 1: Formal parameters, actual parameters and the stack
- ✓ MCQ 1
- ✓ MCQ 2
- ✓ MCQ 3
- ✓ MCQ 4
- ✓ MCQ 5
- ✓ MCQ 6
- ✓ MCQ 7

- A single chapter is usually composed by:
  - The reader of the corresponding lecture
  - A series of exercises which are a combination of:
    - Multiple Choice Questions (MCQ)
    - Forward Assignments (FA)
    - Backward Assignments (BA)

- During the practicums, the teachers will show you more in detail how to solve the Forward and Backward assignments.

- In short, a **Forward Assignment** shows you a program and the (sometimes incomplete) state associated to certain steps of the execution of such program (marked with red blocks to the left of the code). To solve a FA, you need to insert the missing values of variables in *all* incomplete states (remember to click “Next” until the last state is reached). For example:



Welcome (student), abbam@hr.nl

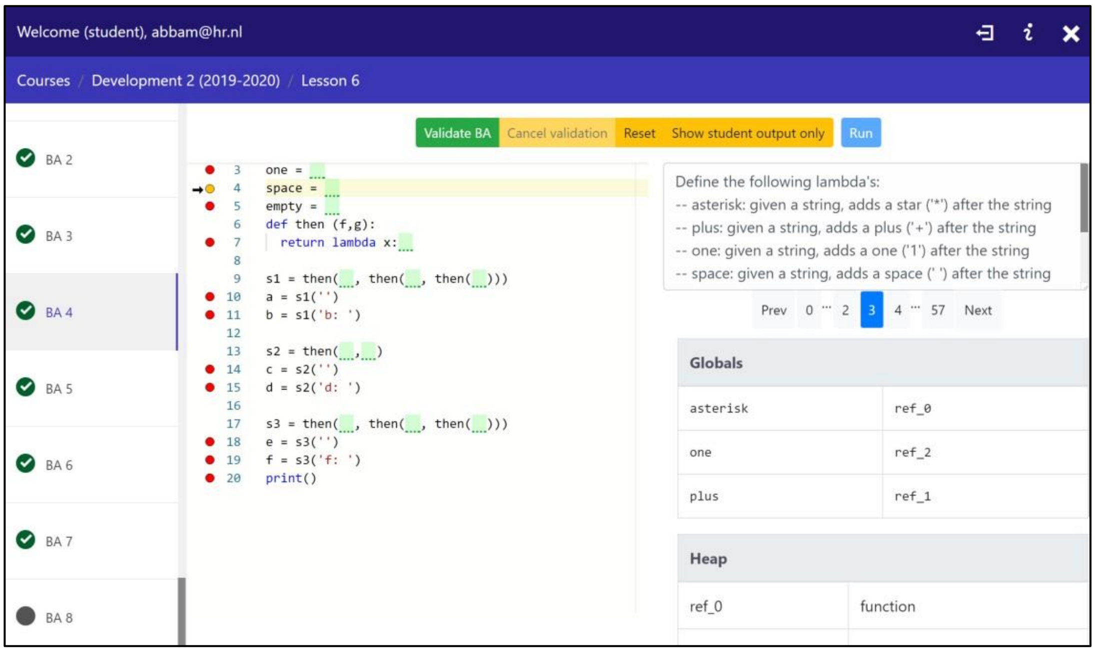
Courses / Development 2 (2019-2020) / Lesson 2

- ✓ MCQ 52
- FA 1
- FA 2
- FA 3
- FA 4

```
def swap (a, b):
  t = a
  a = b
  b = t
  print()
a = 10
b = 20
swap(a, b)
print()
```

The state on the right (Globals, etc.) corresponds to the state of the program when the line of code marked with a yellow block is about to be executed (in the example above, when line 5 is about to be executed).

A **Backward assignment**, instead, shows you an incomplete program and the states associated to some steps of the execution of the complete program (again, marked by red/yellow blocks to the left of the code). By looking at such states, you should be able to fill in the missing parts of the program. For example:


Welcome (student), abbam@hr.nl
Courses / Development 2 (2019-2020) / Lesson 6
BA 2
BA 3
BA 4
BA 5
BA 6
BA 7
BA 8
Validate BA
Cancel validation
Reset
Show student output only
Run
Prev 0 ··· 2 3 4 ··· 57 Next

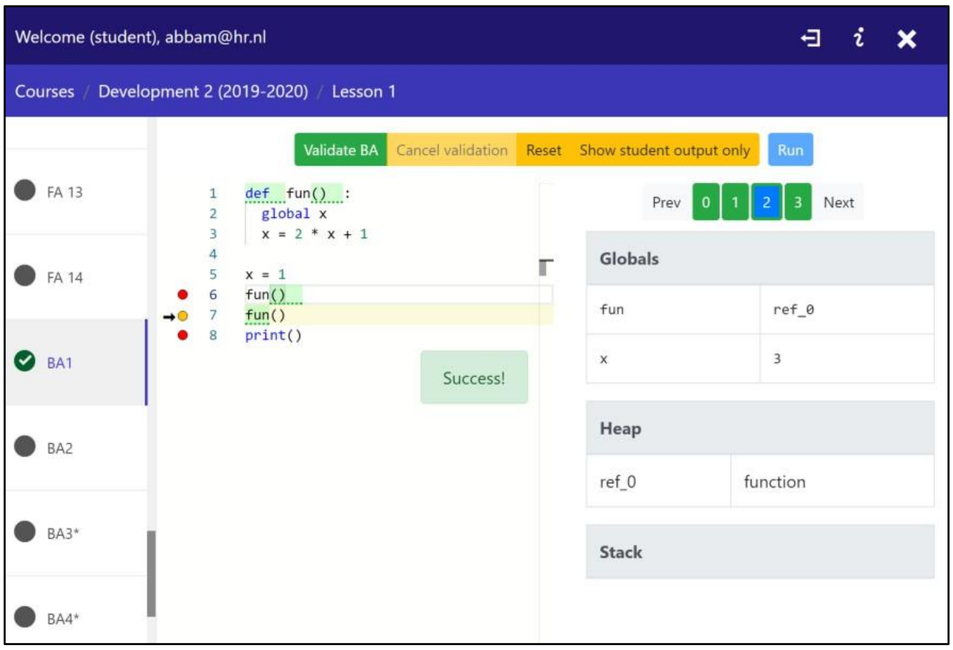
```
one = ....
space = ....
empty = ....
def then (f,g):
  return lambda x:....

s1 = then(...., then(...., then(....)))
a = s1('')
b = s1('b: ')

s2 = then(....,....)
c = s2('')
d = s2('d: ')

s3 = then(...., then(...., then(....)))
e = s3('')
f = s3('f: ')
print()
```

Define the following lambda's:
-- asterisk: given a string, adds a star ('*') after the string
-- plus: given a string, adds a plus ('+') after the string
-- one: given a string, adds a one ('1') after the string
-- space: given a string, adds a space (' ') after the string

Globals	
asterisk	ref_0
one	ref_2
plus	ref_1

Heap	
ref_0	function

To see if your code solves the BA, click on “Validate BA” and you will get feedback.
When an assignment is correctly solved (both FA and BA) a “Success!” green message will appear on screen:



```
def fun():
    global x
    x = 2 * x + 1

x = 1
fun()
fun()
print()
```

Success!

Globals

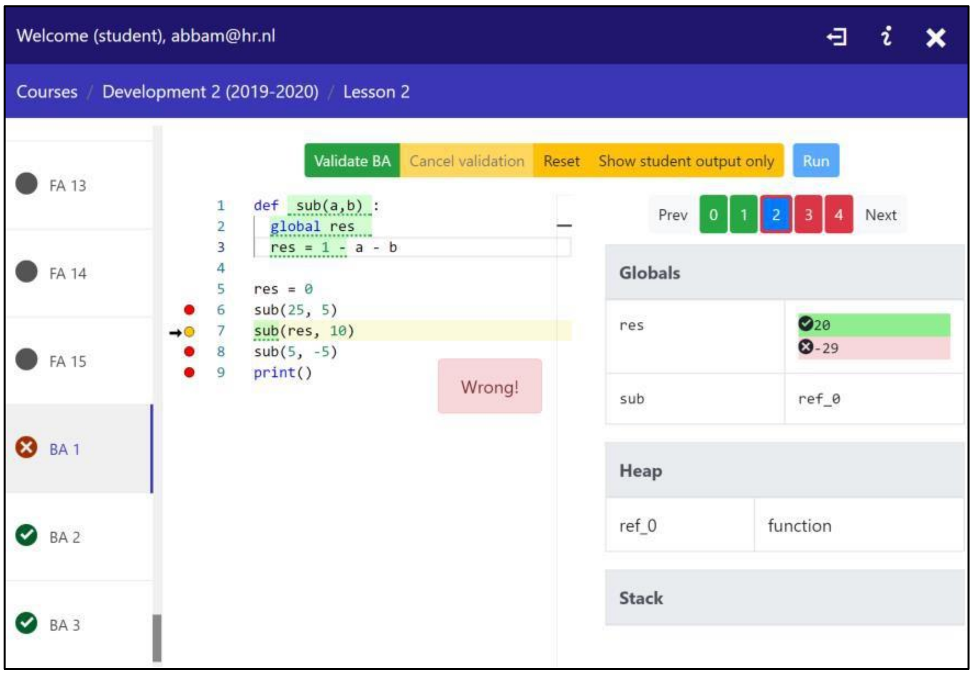
fun	ref_0
x	3

Heap

ref_0	function

Stack

Otherwise, a “Wrong!” red message appears (and in BAs the wrong values of your program are shown in red close to the correct ones in green in the state):



```
def sub(a,b):
    global res
    res = 1 - a - b

res = 0
sub(25, 5)
sub(res, 10)
sub(5, -5)
print()
```

Wrong!

Globals

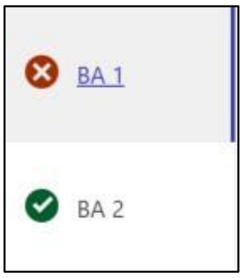
res	✓20 ✗-29
sub	ref_0

Heap

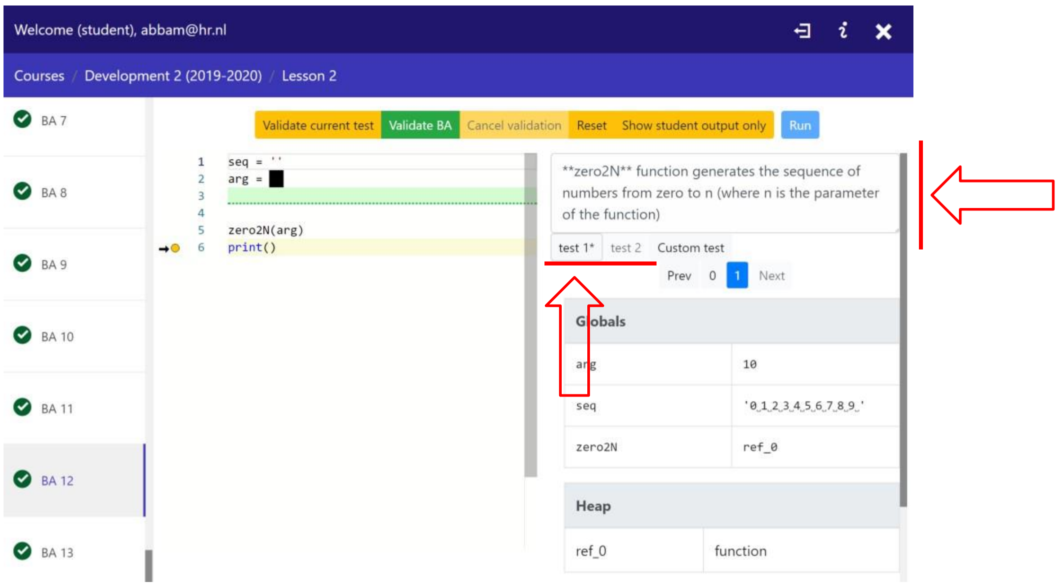
ref_0	function

Stack

The round icon close to the assignment name in the left column also gets such color (red for incomplete/wrong and green for complete):



A **Unit Test BA assignment**, is like a **BA** but, instead of validating your program against one input test, it validates your assignment against a series of tests. Some of these tests might be invisible to challenge you to think of a general solution for the assignment. In order to help you solving the code we also provide you with an optional description that tells you what the program is supposed to do.



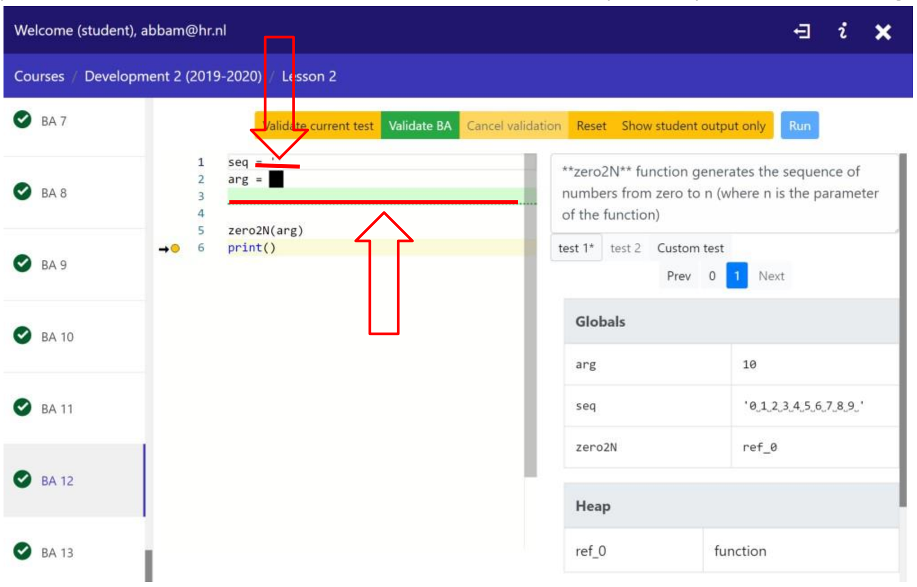
```
seq = ''
arg = ■


zero2N(arg)
print()
```

zero2N function generates the sequence of numbers from zero to n (where n is the parameter of the function)

Globals	
arg	10
seq	'0_1_2_3_4_5_6_7_8_9_'
zero2N	ref_0

Heap	
ref_0	function

The black boxes in code indicate that part of the code is related specifically to the current test; therefore, you do not need to fill it in. As in the standard **BA**'s you only need to fill the green boxes.



Welcome (student), abbam@hr.nl
Courses / Development 2 (2019-2020) / Lesson 2
BA 7 BA 8 BA 9 BA 10 BA 11 BA 12 BA 13
Validate current test · Validate BA · Cancel validation · Reset · Show student output only · Run

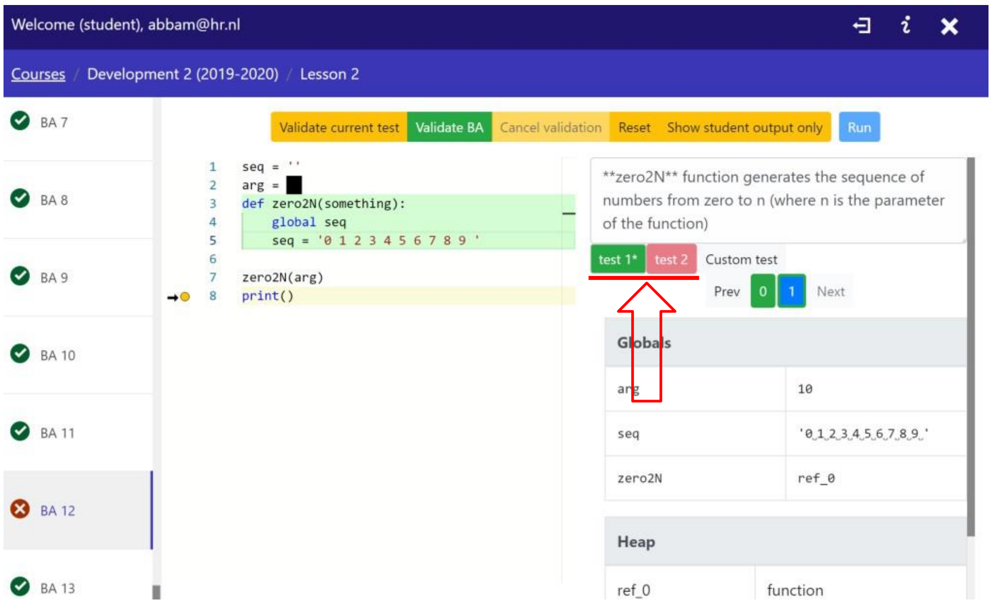
```
seq = ''
arg = ■


zero2N(arg)
print()
```

zero2N function generates the sequence of numbers from zero to n (where n is the parameter of the function)

test 1* · test 2 · Custom test · Prev 0 1 Next

Globals: arg 10; seq '0_1_2_3_4_5_6_7_8_9_'; zero2N ref_0
Heap: ref_0 function

When pressing “Validate BA” each test is validated against your input. Wrong tests will turn red (click on a wrong test to check your mistakes).



```
seq = ''
arg = █
def zero2N(something):
    global seq
    seq = '0 1 2 3 4 5 6 7 8 9 '

zero2N(arg)
print()
```

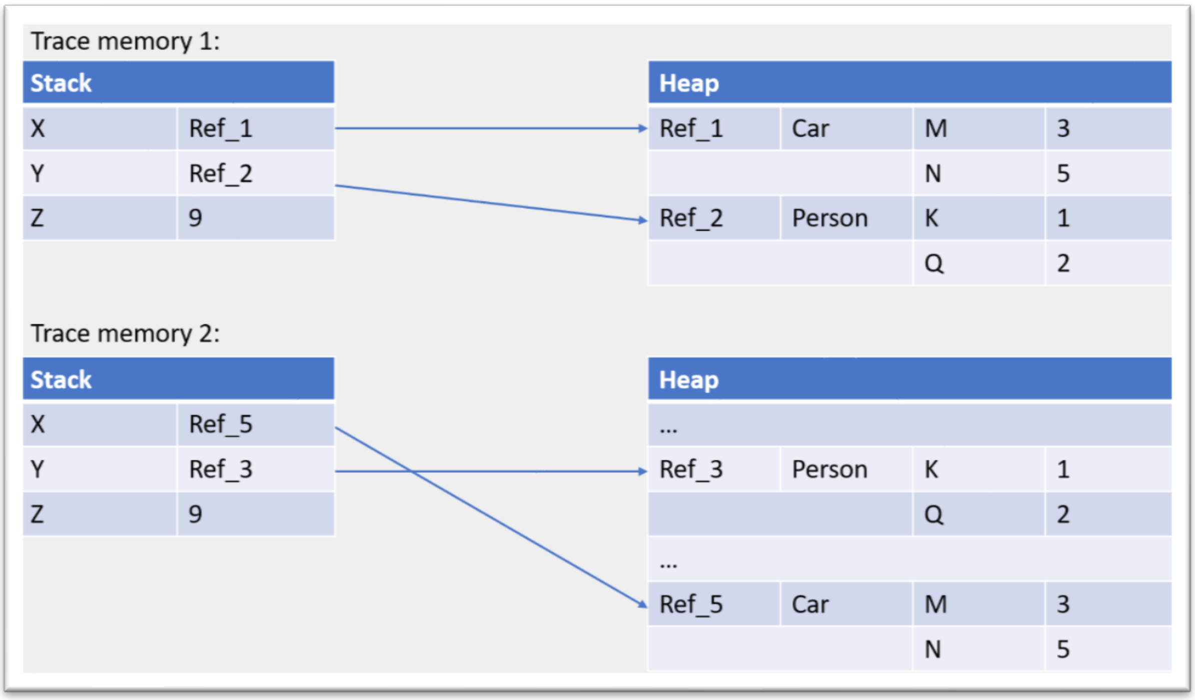
zero2N function generates the sequence of numbers from zero to n (where n is the parameter of the function)

Globals	
arg	10
seq	'0␣1␣2␣3␣4␣5␣6␣7␣8␣9␣'
zero2N	ref_0

Heap	
ref_0	function

Validation strategy

When performing the validation of a backward assignment a structural equality is performed. This means that for each test two memories (the students and the default expected) are tested to have equivalent content. In the following examples of equivalent contents that in GO would successfully validate is presented:



Trace memory 1:

Stack	
X	Ref_1
Y	Ref_2
Z	9

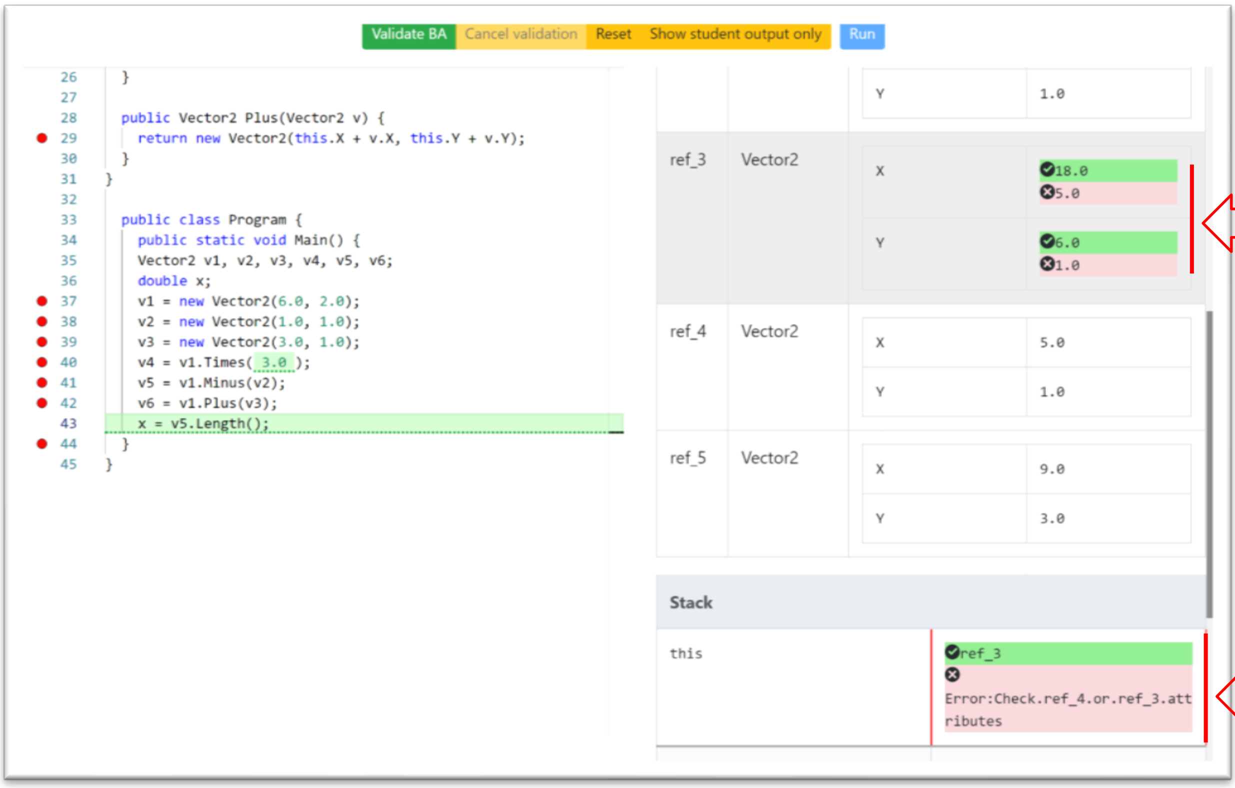
Heap			
Ref_1	Car	M	3
		N	5
Ref_2	Person	K	1
		Q	2

Trace memory 2:

Stack	
X	Ref_5
Y	Ref_3
Z	9

Heap			
...			
Ref_3	Person	K	1
		Q	2
...			
Ref_5	Car	M	3
		N	5

Notice that the test will still validate even if both the references and the ordering in the heap are different. This is because we only check that the referenced content is matching between the two memories. However sometimes the error might be in the reference itself (imagine if we switch ref_1 and ref_2 in the memory trace 1 above); this is obviously a logic error and GrandeOmega cannot distinguish this error from a simple wrong content in the heap. In this case GrandeOmega will indicate both the expected values in the heap and the expected reference.



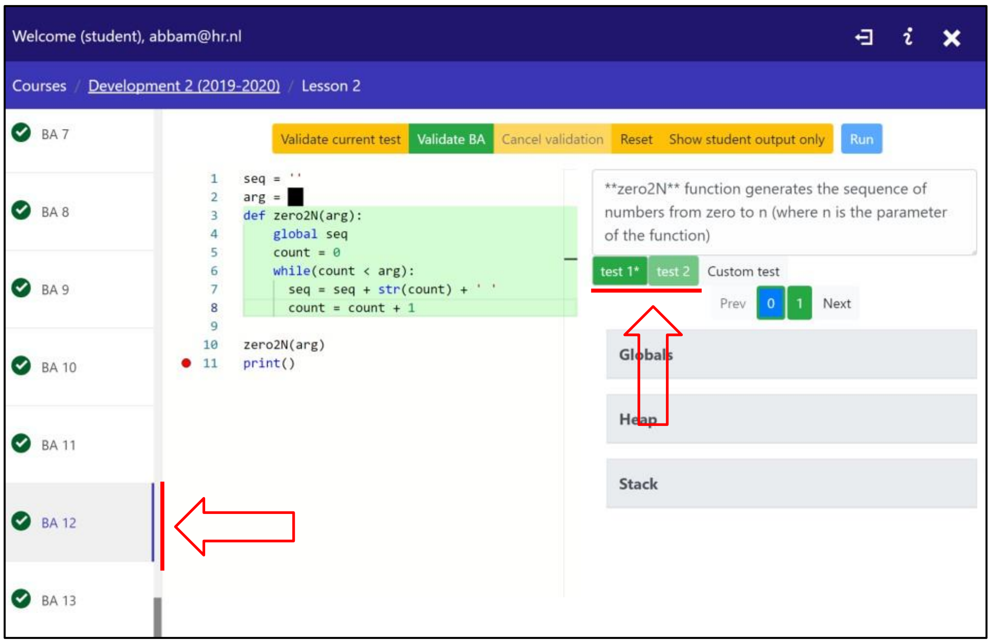
```
    }

    public Vector2 Plus(Vector2 v) {
      return new Vector2(this.X + v.X, this.Y + v.Y);
    }
  }

  public class Program {
    public static void Main() {
      Vector2 v1, v2, v3, v4, v5, v6;
      double x;
      v1 = new Vector2(6.0, 2.0);
      v2 = new Vector2(1.0, 1.0);
      v3 = new Vector2(3.0, 1.0);
      v4 = v1.Times( 3.0 );
      v5 = v1.Minus(v2);
      v6 = v1.Plus(v3);
      x = v5.Length();
    }
  }
```

		Y	1.0
ref_3	Vector2	X	✓18.0 ✗5.0
		Y	✓6.0 ✗1.0
ref_4	Vector2	X	5.0
		Y	1.0
ref_5	Vector2	X	9.0
		Y	3.0

Stack

this	✓ref_3 ✗ Error:Check.ref_4.or.ref_3.attributes

An assignment is considered successful when all tests are passed; i.e. each test button is highlighted with a green color.



```
seq = ''
arg = █
def zero2N(arg):
    global seq
    count = 0
    while(count < arg):
      seq = seq + str(count) + ' '
      count = count + 1

zero2N(arg)
print()
```

zero2N function generates the sequence of numbers from zero to n (where n is the parameter of the function)