{{COURSE_TITLE}}

{{DESCRIPTION}}

UofT ASIC Team

Contents

1 Course path 5

I Course path

2 Getting started 9

3 Lab 01: Starter script 11

1. Course path

{{DESCRIPTION}}

- Who it's for: *one line on who should take this course.*
- Prerequisites: *earlier courses or background, with links.*
- Total time: *about N hours, the sum of the times in the course path.*

This site holds the reading and the lab writeups, and the code that each lab runs is in the course's GitHub repo, under `labs/`. You'll find the other UofT ASIC courses listed at edu.uoftasic.com.

Work down the table in order. A Read row points you to one or more guide pages, which are for reading only, so there's nothing to run. A Do row is a lab, where you run things, make something of your own and check your results.

#		What	Time
1	Read	Getting started	*N* min
2	Do	Lab 01: Starter script. Run the starter Python script, then change the message it prints and predict whether its checksum goes up or down.	*N* min

I Course path

2 **Getting started** 9
2 Course layout 9
2 Lab structure 9
2 Math and figures 9
2 Check yourself 10
2 Answers 10

3 **Lab 01: Starter script** 11
3 Before you start 11
3 Part 1: Set up the lab package 11
3 Part 2: Run the starter 11
3 Part 3: Change the message 12
3 Part 4 (optional): Swap two characters 12
3 Finishing 12
3 Answers 13

2. Getting started

This page explains how the course is laid out and how its pages show math and figures. Read it before Lab 01: Starter script.

Course layout

The course path on the home page lists all the pages in the order you should work through them. Each row is one of two kinds:

- A Read row points to guide pages like this one, which explain the ideas with figures and worked examples. You don't need a terminal open to read them.
- A Do row points to a lab writeup, and the labs are where you run things.

The files a lab runs (code, data, and any scripts that check your results) are in the course's GitHub repo under `labs/`, and this site has only the writeups. You'll run your first script, `labs/lab-01/src/main.py`, in Lab 01, Part 2.

Lab structure

Each lab is split into numbered parts with numbered steps. Most parts open with the question they answer, and the steps then work out the answer. Each part has a "You should see" check, which tells you whether what you did worked. In at least one part you make something yourself, such as editing a file, writing a test, or working out a number before a script or tool prints it.

Some steps ask you to write down what you expect before you run something, so that you have a prediction of your own to compare the result with.

When a step asks a question, its answer is in a collapsed block right under it. Try to answer the question yourself first, and then click "Answer" to open the block. (In the PDF version of a page, the answers are collected at the end of the page instead, and a short note marks where each one was.) The questions in Check yourself at the end of this page work the same way.

A part marked "(optional)" comes after all the required parts. No later part needs it, so you can skip it, and the lab's time estimate doesn't include it.

Every lab ends with a Finishing section. It tells you how you'll know you're done, asks any questions that you can answer from your own work, and says what to submit, if anything. The last line of each lab points to the next row on the course path.

Math and figures

Math on these pages is typeset with MathJax on the website and by Typst in the PDFs. Inline math sits inside the sentence, like $y[n] = x[n] * h[n]$ for discrete convolution, and display math gets a line of its own:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]\, h[n-k] \tag{2.1}$$

Figures look like this:



Figure 2.1: Sample figure

Check yourself

1. Which kind of course path row has you run commands?

   Answer: A1, in Answers at the end.

2. Where are the files that a lab runs?

   Answer: A2, in Answers at the end.

Answers

A1. Check yourself
A Do row, which points to a lab writeup. The labs are where you run things.
A2. Check yourself
They're in the course's GitHub repo, under `labs/`. This site has only the writeups.

3. Lab 01: Starter script

You'll run a starter Python script, then change the message it prints and predict what happens to its checksum. The script and the other files for this lab are in `labs/lab-01/` in the course repo.

Before you start

- Read Getting started.
- You need git, Python 3.10 or newer, and a clone of the course repo:

```
git clone https://github.com/uoftasic/{{COURSE_ID}}.git
```

- This starter needs only Python. Labs that use the EDA tools (IIC-OSIC-TOOLS) run in the shared workspace that you set up in IC101.

Time: about *N* minutes, plus *N* minutes for the optional Part 4.

Part 1: Set up the lab package

1. From the root of your clone, go to the lab folder:

```
cd labs/lab-01
```

2. Install the lab's Python dependencies:

```
python3 -m pip install -r requirements.txt
```

If pip stops with `error: externally-managed-environment`, your system Python doesn't let pip install into it (the system Python on Ubuntu 24.04 does this). Make a virtual environment in the lab folder and run the install inside it:

```
sudo apt install python3-venv   # Ubuntu only: python3 -m venv needs this package
python3 -m venv .venv
. .venv/bin/activate
python3 -m pip install -r requirements.txt
```

Activating the virtual environment changes only the terminal you ran it in, so if you open a new terminal later in the lab, run `. .venv/bin/activate` in that one too.

You should see pip finish without an error. It won't install anything, because the starter's `requirements.txt` has no packages in it.

Part 2: Run the starter

What does the starter print, and where does its checksum come from?

1. Open `src/main.py`. Which function works out the number the script prints after `checksum=`, and how?

Answer: A1, in Answers at the end.

2. Run the script:

```
python3 src/main.py
```

You should see:

```
lab-01 ready
checksum=13
```

Part 3: Change the message

What happens to the checksum when the message gets shorter?

1. In `src/main.py`, change `ready` to `done` in the line `message = "lab-01 ready"`. Keep the four spaces at the start of the line, or Python stops with an `IndentationError`. The line should now read:

   ```
       message = "lab-01 done"
   ```

2. `done` is one character shorter than `ready`. Before you run the script, write down whether you expect the new checksum to be bigger or smaller than 13.
3. Run the script again:

   ```
   python3 src/main.py
   ```

   You should see:

   ```
   lab-01 done
   checksum=899
   ```

4. Compare the result with your prediction. The message got shorter, so why did the checksum go up?

   Answer: A2, in Answers at the end.

Part 4 (optional): Swap two characters

Can this checksum tell when two characters swap places? No later part depends on this one, so you can skip it.

1. Your Part 3 message with the `0` and `1` swapped is `lab-10 done`. Before you run anything, write down what you expect its checksum to be.
2. From `labs/lab-01`, run `checksum()` on the swapped message. The `PYTHONPATH=src` at the start of the command tells Python to look in `src/` for modules, which is how it finds `main.py`.

   ```
   PYTHONPATH=src python3 -c 'from main import checksum; print(checksum("lab-10 done"))'
   ```

   You should see `899`, the same checksum as `lab-01 done` in Part 3.
3. Why didn't the checksum change?

   Answer: A3, in Answers at the end.

Finishing

You're done when `python3 src/main.py` prints `lab-01 done` and `checksum=899`, from your edit in Part 3.

Questions

1. Your Part 3 message gives `checksum=899`. If you added one more character to the end of the message, what character code would it need for the checksum to come out as 0?

   Answer: A4, in Answers at the end.

Submit

Nothing to submit.

This starter doesn't write any files. When a lab does write large outputs, such as GDS files or waveform dumps, keep them in its `results/` folder (here `labs/lab-01/results/`) and out of `docs/`. Use Git LFS if you need to share them.

Next on the Course path: *the next row, with links to its guide pages or lab.*

Answers

A1. Part 2: Run the starter

The `checksum()` function does. It adds up the character code of each character in the message (`ord(c)`), then divides the total by 997 and keeps the remainder (`% 997`).

A2. Part 3: Change the message

The character codes of `lab-01 ready` add up to 1010, and those of `lab-01 done` add up to 899. Because 1010 is more than 997, the old checksum was what was left over after dividing by 997, which is $1010 - 997 = 13$. The new total of 899 is less than 997, so dividing leaves all of it as the remainder, and the checksum is 899 itself.

A3. Part 4 (optional): Swap two characters

Addition doesn't depend on order, so the same characters in any order add up to the same total and give the same checksum. A checksum that only adds character codes can't catch two characters swapping places.

A4. Finishing

It would need code 98. The total then becomes $899 + 98 = 997$, and 997 divided by 997 leaves a remainder of 0. Code 98 is `b`, so `message = "lab-01 doneb"` prints `checksum=0`.