Lesson 2: Crunching numbers

**Enjoyed teaching these lessons? Found a mistake? Share feedback at** [**the-cc.io/feedback**](http://the-cc.io/feedback)**.**

## Introduction

In the previous lesson, learners were introduced to displaying messages, assigning values to variables, and receiving input from the keyboard. This lesson will help them gain a deeper understanding of assignments, and will explicitly address some of the common misconceptions around the semantics of assignment statements.

Learners will also be introduced to using arithmetic expressions and receiving numerical input from the keyboard. These are two key components that will allow them to progress to building more elaborate programs in the lessons to follow.

The main activity in this lesson will require learners to construct their own short programs for the first time, through scaffolded tasks.

## Learning objectives

* Describe the semantics of assignment statements
* Use simple arithmetic expressions in assignment statements to calculate values
* Receive input from the keyboard and convert it to a numerical value

## Key vocabulary

Input, output, variables, operators, expressions, integer and string type, execution, walk-through

## Preparation

**Subject knowledge:**

* You will need to be familiar with using a Python IDE.
* You will need to be able to locate and correct syntax errors in Python programs.
* You will need to be comfortable with the use of output, input, and assignment in Python, including arithmetic input.
* You will need to be comfortable with the use of arithmetic operators and expressions.
* You will need to be aware of common misconceptions that you may encounter with novice learners. See the [common misconceptions](#_22t23np7msw5) in the ‘Notes on pedagogy’ section for a list of misconceptions relevant to this lesson.

**You will need:**

* Slides — note that some slides contain animations
* Activities:
  + Order matters: worksheet, solutions, and [Python code starting point](https://the-cc.io/py-order-20) (the-cc.io/py-order-20)
  + How to input numbers: worksheet and [Python code starting point](https://the-cc.io/py-moon-20) (the-cc.io/py-moon-20)
* Homework and solutions
* A Python interpreter and IDE — we suggest using the Mu editor ([codewith.mu](https://codewith.mu/)), or an online environment such as [Repl.it](https://repl.it/); if you are using the Mu editor, then you might find the following guide useful to help you prepare: [Getting started with Mu](http://rpf.io/mu) (rpf.io/mu)

**You may need:**

* Additional Python code for the ‘Order matters’ activity:
  + [Tracing example](https://the-cc.io/py-days-2) (the-cc.io/py-days-2)
  + [Solution](https://the-cc.io/py-order-21) (the-cc.io/py-order-21)
* Additional Python code for the ‘Age calculation’ activity:
  + Live coding: [Age calculation, without int](https://the-cc.io/py-age-20) (the-cc.io/py-age-20)
  + Live coding: [Age calculation, with int](https://the-cc.io/py-age-21) (the-cc.io/py-age-21)
* Additional Python code for the ‘How to input numbers’ activity:
  + [Weight on the moon — solution](https://the-cc.io/py-moon-21) (the-cc.io/py-moon-21)
  + [Dog years — solution](https://the-cc.io/py-dogyears-21) (the-cc.io/py-dogyears-21)
  + [Weight on the moon (explorer task) — solution](https://the-cc.io/py-moon-22) (the-cc.io/py-moon-22)
  + [Age calculation (explorer task) — solution](https://the-cc.io/py-age-22) (the-cc.io/py-age-22)
* Python cheat sheets on output, assignment, input, operators, and expressions

## Assessment opportunities

Multiple choice questions throughout the lesson provide a means for quick formative assessment on specific concepts. You can also assess learners’ answers to the worksheets. In addition, you can assess learners through observation, for example, by assessing how learners interact through pair programming and collaborate to solve problems. The homework will also provide an opportunity to assess whether the lesson objectives have been achieved.

## 

## 

## At a glance

| **Starter activity**  10 mins | **Make predictions**  Use three multiple choice questions on short code samples to help learners recall some of the most important points from the previous lesson and connect those points with ideas that will be introduced in this lesson. |
| --- | --- |
| **Activity 1**  10 mins | **Assignment and expressions**  Explain the semantics of assignment statements that include arithmetic expressions through a sequence of examples. Perform a walk-through of a short sequential program in detail, to illustrate the mechanics of assignment and expression evaluation during program execution. Finally, ask learners to work through a Parson’s Problem. |
| **Activity 2**  5 mins | **Subtle points**  Pose and discuss two multiple choice questions that address common misconceptions about assignment statements. |
| **Activity 3**  8 mins | **Age calculation**  Perform a live coding session to illustrate how to create programs that receive numerical input. |
| **Activity 4**  20 mins  7–8 mins per task  4 mins solutions | **Programming tasks**  Hand out the ‘How to input numbers’ worksheet and ask learners to apply what they have learnt in programming tasks that involve numerical input and arithmetic expressions. |

## 

## Outline plan

Please note that the slide deck labels the activities in the top right-hand corner to help you navigate the lesson.

*\*Timings are rough guides*

| **Starter activity**  (Slides 2–4)  10 mins | **Make predictions**  The goal of this starter activity is to help learners recall some of the most important points from the previous lesson, but also connect those points with ideas that will be introduced in this lesson.  The starter activity involves three similar pieces of code, each accompanied by one or two closed-form questions (multiple choice and true/false). For each of these pieces of code, ask learners to take a few seconds to reflect on their answers independently, then to turn to a classmate and discuss them in pairs (‘think, pair, share’). Aggregate the learners’ responses, ask for explanations if necessary, and click through the animations on the slides to reveal the correct answers.  **Code fragments #1 and #2**  The lines of code on slides 2 and 3 are identical, but they have been swapped around. This will provide you with an opportunity to emphasise how instructions are executed in order, and that a variable needs to have been assigned a value before that value is referenced.  **Code fragment #3**  The piece of code on slide 4 involves user input. This will allow you to stress that the actual value of the user variable is not known while the programmer is writing the program. That value is determined by the user during program execution. However, the programmer is still able to reference that unknown value in the program, using the name of the variable.  Explain to learners that the output of the program can be different every time it is executed, even though the exact same instructions are executed: what is different between executions is the value of the user variable. |
| --- | --- |
| **Activity 1** (Slide 6–13)  10 mins | **Assignment and expressions**  In the previous lesson, learners were introduced to examples of very simple assignments, where a numerical or string literal was directly assigned to a variable. In this activity, they will be provided with a sequence of assignment examples, which will include expressions and will introduce the semantics of assignment statements.  **Code fragment #1: Simple assignment**  The example on slide 6 involves a simple assignment, like the ones that learners are familiar with. The number of days in a year is assigned to a variable. Stress that assignments are not equations, they are instructions to be carried out.  **Code fragment #2: Assignment that involves an expression**  The example on slide 7 is a refinement of the first, where the days in a year are computed using an expression. Describe the semantics of an assignment statement: **first** compute the value of the expression on the right, and **then** assign the computed value to the variable on the left. This suggests that assignment statements should be read from right to left, in order for them to be understood correctly.  Slide 8 shows the arithmetic operators that can be used to form arithmetic expressions in Python. This is intended to be used as a reference, so read through the slide briefly, simply listing the available operators to convey an idea of the sort of calculations that can be performed.  At this point, you may wish to remind learners of operator precedence. They will be familiar with it from mathematics, but it is often a source of confusion.  **Code fragment #3: Expressions that reference variables**  In the example on slide 9, the number of days in four years (a quadrennium) is computed using the number of days in a single year. There are two variables involved, and the value computed for one depends on the value of the other. This provides another opportunity to discuss how it is necessary to assign a value to a variable before that variable is referenced.  **Program execution**  Perform a step-by-step walk-through of this program, along with your learners (slides 10–12). This practice will help them develop an initial model of a **notional machine** (to find out more, see the ‘Notes on pedagogy’ section): it will help them to understand how instructions are executed, how expressions are evaluated, how values are assigned to variables, etc.  The slides include a dual description of each instruction as it is executed. One part of the description focuses on how the instruction is executed (like a translation from syntax to operational semantics), while the other explains the function, i.e. what the purpose of the instruction is. This is directly influenced by the **block model** (to find out more, see the ‘Notes on pedagogy’ section).  The slides also illustrate how to use **sketching** to keep track of the values of variables.  **Note:** Some of the slides contain the Scratch blocks that correspond to (relevant parts of) the Python program, so that learners can make associations with concepts that they are already familiar with. The online course *Scratch to Python: Moving from block-based to text-based programming* includes [a step that draws parallels between Scratch and Python syntax](http://rpf.io/scratchtopython), as well as a relevant cheat sheet at the end.  **Order matters (worksheet)**  Hand out the ‘Order matters’ worksheet and ask learners to carry out the task, which is essentially a Parson’s Problem (to find out more, see the ‘Notes on pedagogy’ section). |
| **Activity 2**  (Slides 14–15)  5 mins | **Subtle points**  This short activity calls on learners to answer two multiple choice questions that address common misconceptions about assignment statements. Answers and explanations are provided on the slides.  **MCQ1:** What will be the value of double, after executing line (A)?  number = 5  double = 2 \* number  number = 15 line labelled (A)  When learners interpret assignments as equations, they believe that the value of double is ‘updated’ to 30 as soon as the value of number becomes 15 (answer 2). They may also believe that it is not possible or valid to assign a new value to number (answer 3).  **MCQ2:** What will be the value of number, after executing line (A)?  number = 5  number = number + 10 line labelled (A)  When learners interpret assignments as equations, they believe that there can be no valid value for number that satisfies the ‘equation’ in line (A) (answer 3). Other common misconceptions include that it is not possible or valid to assign a new value to number (answer 4) or that it is possible for a variable to refer to more than one value at the same time (answer 1). |
| **Activity 3**  (Slides 16–19)  8 mins | **Age calculation**  In order for learners to progress to building more meaningful programs, they need to know how to receive numerical input from the keyboard. Surprisingly, this may be (syntactically) challenging in Python.  In this activity, you will use **live coding** (to find out more, see the ‘Notes on pedagogy’ section) to guide learners to develop a simple program that asks the user for their year of birth and calculates their age. This will allow you to demonstrate how to develop a solution using the constructs that they are familiar with, but also manage their reactions when they realise that the program will not function properly because the input received from the keyboard is textual, while the input required by the program is numerical.  Pair learners, as they will be doing **pair programming** for the rest of the lesson.  **Live coding: Input**  Inform learners that you will be making a program that asks the user for their year of birth and calculates their age. Explain why you need to use a variable to refer to the user’s year of birth and ask how the program can receive user input.  This is what you should have by the end of this step:  print("Year of birth?")  birth\_year = input()  Make sure that you demonstrate running the program to check that everything works so far, even though it’s only two lines long.  **Note:** At this point, learners are not familiar with using the int function to convert the string that input returns to a number. You will introduce this later on in this activity. Therefore, for now, you will only use input, but you will expand on this later on.  **Live coding: Processing the input**  Discuss with learners what sort of expression you would need to use in your program to calculate the user’s age. When you reach a conclusion, type in the expression, along with print, to output the result.  print("Year of birth?")  birth\_year = input()  age = 2020 - birth\_year  print("You are", age, "years old")  **Live coding: Numerical input**  Run the program. You will be presented with a TypeError. Display slide 18 and explain that the reason for this error is that the program is trying to compute the difference between a number (the current year) and a piece of text (the year of birth).  Use int to convert the year of birth to an integer (slide 19) and make sure that learners are also able to do this in their programs. Stress that this is what they will need to do whenever they are building any program that requires integer input. |
| **Activity 4**  (Slides 20–22)  20 mins  7–8 mins per task  4 mins solutions | **Programming tasks**  Hand out the ‘How to input numbers’ worksheet and ask learners to work on the programming tasks. The worksheet includes the age program that you just created, as a worked example for learners to use as a reference.  Remind learners to switch their pair programming roles every time they complete a task.  Conclude the lesson by providing solutions to the tasks, and discussing any questions that the learners may have. |
| **Homework** | Assign the homework for this lesson. You could also include questions from the assessment that learners will complete at the end of the unit. |

## Notes on pedagogy

### Common misconceptions

Throughout these lessons, you should be aware of the common misconceptions that may arise with novice programmers. The misconceptions below are a selection from Appendix A of Juha Sorva’s PhD thesis, *Visual Program Simulation in Introductory Programming Education*, and are relevant to this lesson. The list will help you avoid using any examples or explanations that may give rise to these misconceptions, and it will also help you spot them, should they arise with your learners.

**M1, 2, 7:** The computer knows or is able to deduce the intention of the program or of a piece of code, and acts accordingly. The machine understands English.

**M4:** The system does not allow unreasonable operations.

**M6:** Difficulties with telling apart the static and dynamic aspects of programs.

**M23:** Difficulties in understanding the sequentiality of statements.

**M3:** Values are updated automatically according to a logical context.

**M150:** Difficulties understanding the effect of input function calls on execution.

**M155:** Numbers are just numbers. (Why have int and float separately?)

**M9:** A variable can hold multiple values at a time / ‘remembers’ old values.

**M10:** Variables always receive a particular default value upon creation.

**M158, 159:** Confusion between data in memory and data on screen. The computer keeps what has been printed in memory (as part of state?).

**M11:** Primitive assignment works in the opposite direction.

**M12:** Primitive assignment works both directions (swaps).

**M16:** Assignment moves a value from a variable to another.

**M13, 15:** Limited understanding of expressions which lacks the concept of evaluation. Primitive assignment stores equations or unresolved expressions.

### Parson’s Problems

The online course [Programming Pedagogy in Secondary Schools: Inspiring Computing Teaching](http://rpf.io/secondarypedagogy) by the Raspberry Pi Foundation describes Parson’s Problems as “a task that involves giving learners all of the lines of code required to solve a problem, but with the lines jumbled so that they are not in the correct order. Learners are required to place the lines into the correct order to form a working code segment. The main benefit of Parson’s Problems is that the learner is focusing on code structure rather than syntax. The process lowers the cognitive load, allowing learners to practise sequencing and the meaning of the code”.

### Live coding

Greg Wilson, in his book [*Teaching Tech Together*](https://teachtogether.tech/), describes live coding as follows: “the teacher writes code in front of the class while the learners follow along, typing it in and running it as they go”. He calls it “the most effective way to teach programming” and goes on to list advantages and cite relevant research.

Live coding should not be improvised. It is a planned, well-structured ‘performance’, but it also provides teachers with the opportunity to actively address the unanticipated. You can find a relevant discussion and a detailed example in the online course [Programming Pedagogy in Secondary Schools: Inspiring Computing Teaching](http://rpf.io/secondarypedagogy).

### Worked examples

### The worksheets handed out to learners throughout this unit will often start with a worked example, i.e. an annotated solution to a problem. The tasks that follow the worked example will be closely linked to it, so that learners can use it as a reference point. These worked examples will also have been presented in class, using live coding, so the reasoning behind the solution will have been fully explained to learners.

Worked examples reduce cognitive load and can help learners assimilate new information. You can find out more about them in the resources referenced at the end of the lesson plan.

### Multiple choice questions and misconceptions

### A lot of the multiple choice questions used within the lessons are targeted at specific misconceptions, which are outlined in the lesson plan. The same applies for the questions suggested for assessment, with the rationale behind the questions explained in the accompanying solutions.

Exploring the answers to these questions in class in a structured manner that involves group and class discussion (using ‘think, pair, share’ in this case) links to a teaching method called ‘peer instruction’. There is evidence to suggest that this method improves retention, as well as promoting inclusivity. You can find out more about peer instruction in the resources referenced at the end of the lesson plan, so that you can make the most of these questions and the discussions that follow them.

### 

### Sketching, walk-throughs, and trace tables

The notion of ‘sketching’ variables and their values is referenced in the lesson slides. This links to the idea of using drawings to illustrate how program state (e.g. the values of variables, and the contents of data structures) is modified during program execution. This can provide a visual alternative to trace tables that may be less cumbersome for learners at this stage.

A very useful tool for step-by-step walk-throughs of Python programs and visualisation of program state is [Python Tutor](http://pythontutor.com/visualize.html) (pythontutor.com/visualize.html).

The first lesson in the Y7 programming unit contains an Unplugged activity called ‘Sequence the sounds’ that you can use to reinforce the idea that program statements are executed in sequence.

### Notional machine

The notional machine, introduced by du Boulay in 1986, refers to “the general properties of the machine that one is learning to control”. It is an abstraction that aims to explain program execution and is linked to the language used to create programs. Learners build models of the notional machine as they are learning to program. These lessons aim to provide a structured and detailed description of what goes on during program execution, with the purpose of supporting students to build accurate models of the notional machine.

### The Block Model

Schulte’s [Block Model](https://www.researchgate.net/publication/247927531_Block_Model_an_educational_model_of_program_comprehension_as_a_tool_for_a_scholarly_approach_to_teaching) is “an educational model of program comprehension” that describes understanding and learning programming using three dimensions and four levels.

The three dimensions are: text surface, program execution, and function (purpose). In these lessons, especially while tracing through a program being executed, you will link these three dimensions by describing how a static program turns into a dynamic process when executed, along with the purpose of execution, i.e. what the instructions of the program aim to achieve.

## Additional sources

* Pedagogy Quick Read: [Live coding](https://the-cc/qr05)
* Pedagogy Quick Read: [Worked examples](https://the-cc/qr02)
* Pedagogy Quick Read: [Peer instruction](https://the-cc/qr04)
* [Programming Pedagogy in Secondary Schools: Inspiring Computing Teaching](http://rpf.io/secondarypedagogy) on FutureLearn, which contains sections on Parson’s Problems, worked examples, and live coding
* Greg Wilson’s book [*Teaching Tech Together*](https://teachtogether.tech/), which contains sections on [live coding](https://teachtogether.tech/#s:performance-live) and [notional machines](https://teachtogether.tech/#s:models-notional). There is also a short description of Parson’s Problems in the section on [cognitive load](https://teachtogether.tech/#s:architecture-load).
* [Scratch to Python: Moving from Block- to Text-based Programming](http://rpf.io/scratchtopython) on FutureLearn

Resources are updated regularly - the latest version is available at: [the-cc.io/curriculum](http://the-cc.io/curriculum).



This resource is licensed by the [Raspberry Pi Foundation](https://www.raspberrypi.org/) under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International licence. To view a copy of this license, visit, see [creativecommons.org/licenses/by-nc-sa/4.0/](https://creativecommons.org/licenses/by-nc-sa/4.0/).