

Exploring the Python side of Code Jumper:   
 CJ Threads

# Objectives

Students will:

* Explore the similarities and differences between block coding in the CJ app and the Python version of the same program
* Demonstrate understanding of the CJ Threads user interface

# Expected Outcomes

* All students will have a familiarity with the Python side of the app
* Most students will begin to see, hear, or feel differences between Code Jumper block coding and CJ Threads Python code
* Students will start to think of new ways of writing block code in Python code

# Key Vocabulary

**Toolbox** – Shortcut buttons for specific items in Code Jumper, such as *Play*, *Pause*, *Loop*, *If*, *Else*, *Thread*, *Variable*, *Comment*, *Edit,* and *Delete*. At the bottom of this section are indent options, with a button to *Indent Left* and another to *Indent Right*.

**Code Window** – Editable lines of code that can be modified using the buttons in the toolbox or the designated hotkeys listed in the hotkeys document. A new workspace displays code in the Python programming language that was previously shown in the original block code window.

**Output Window** –Shows the user the results of their program or any errors they may have when running a program

**Program** **Buttons** – *Home*, *Line Numbers* (toggle button: line numbers can be toggled on or off), *Save*, *Load*, *Export*, *Run*, *Stop*

**Home (block side of code)** –The app screen showing the Code Jumper block code

**CJ Threads (Python side of code)** – The app screen showing the Python code

**Debugging** – When a userlooks for mistakes (called “bugs”) and fixes them, so the code works correctly

# Building Blocks of Python

**Comments** – Comments are written to explain sections of code or highlight valuable information. In Python, any line that starts with a hashtag *#* (number sign) will denote a comment. The first line on CJ Threads programs begins with a comment that can be edited to display the name of the program.

**Import** – The Import command allows users to import modules, classes, or functions into a Python program to expand the functionality of the code. In CJ Threads, there are two different import statements:

Import – Imports a module and all its functionality. For example, on line 5, the command ***import soundplayer*** will import the module that allows sounds to be played.

From – From imports, a specific class or function from a module. For example, on line 7, the command ***from random import randrange*** will import a function to create random numbers in a given range.

**Variable** – Variables are symbolic names that refer to numbers, strings, classes, and other types of data. Lines 9–12 initialize (sets the starting value) several variables that are used in CJ Threads programs. For example, Lines 9 and 10 initialize the sound player variable and set the location that stores the sounds used in the app. As another example, Line 11 initializes the sound sets. Finally, Line 12 initializes the variable *X* that can be used in the programs created in CJ Threads. This represents the variable plug.

**Class** – A set of functions and variables that are grouped together to create objects. A class is a template of bundling variables and functions that can be referenced by an object. A class is like a blueprint for a house. Just like a blueprint tells builders how to make a house, a class tells the computer how to create objects. **Object** –An object is an instance of a class. If a class is the blueprint, an object would be the house built according to the blueprint. We can make objects and perform different operations on them. You can personalize your individual house by painting the outside your favorite color and picking different doors. You could even turn your guest bedroom into a CJ-themed recording studio! Even then, your house would share a lot in common with your neighbor’s house.

**Function** –A function is a block of code that can take data and perform certain operations, such as playing audio. A function is like a microwave in a house. Just like in a function to play audio where you set the volume and song, in the microwave you can set the time and power level to heat up your food.

**Threads** –Threads allow multiple blocks of code to run at the same time. In Code Jumper, each thread has its own sound set and will play sounds from that concurrently (which means at the same time)

**Indentation** –Allows a block of code to be placed inside functions, loops, and ***if...else*** statements. Indentation can be written using either tabs or spaces, but in CJ Threads, indentation uses 4 spaces. A new level of indentation will show that the indented block of code is run inside the function, loop, or ***if...else*** statement before it.

For best screen reader practices, set Punctuation to *Speak All* or *Most* when viewing this document so the user is aware of indentations or any key punctuation that is used in Python. If the user is using a refreshable braille display, the braille display will automatically present the correct formatting.

Resources

* Twinkle, Twinkle program card (can be found later in this document)
* Code table
* Code Jumper tutorial videos
  + Code Jumper app: <https://www.youtube.com/watch?v=vg72YPz6CWY>
  + The Hub: [https://www.youtube.com/watch?v=KGb51PW9zJQ&lis](https://www.youtube.com/watch?v=KGb51PW9zJQ)
  + Play and Pause pods: <https://www.youtube.com/watch?v=446jCw8qcDI&t>
* [Python Documentation](https://docs.python.org/3/)

# Activity: Build a Program Using the Code Jumper Pods and Explore in the Python Side of the App

## Materials

* Twinkle, Twinkle code card (can be found later in this document)
* Compare and Contrast table
* Reflection questions

## Instructions

Building the program on the Code Jumper side of the app:

1. Using the code card for Twinkle, Twinkle, build the program on the Code Jumper side of the app. Play the program and using debugging skills, ensure that the program is correct.

# Code Card: Twinkle, Twinkle

 THREAD 1 Twinkle, Twinkle

   PLAY Twinkle 1 for 1 times speed

 PLAY Twinkle 2 for 1 times speed

 PLAY Little for 1 times speed

  PLAY Star for 1 times speed

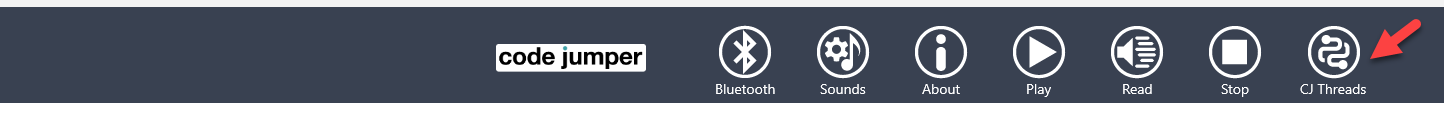
END THREAD





Flip to CJ Threads (the Python side of the app).

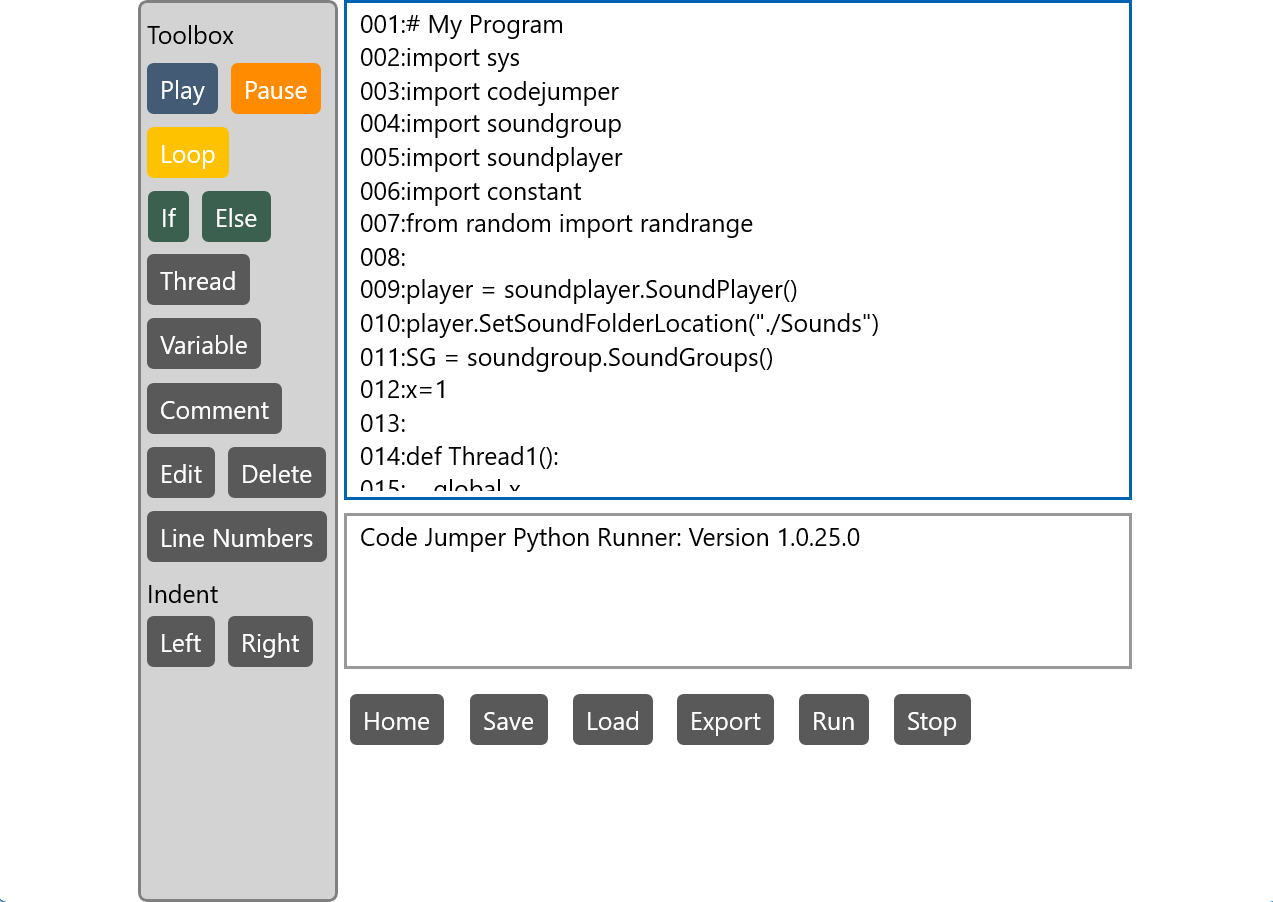
1. Select the *CJ Threads* button in the Code Jumper app toolbar. This will open the Python screen in Code Jumper.

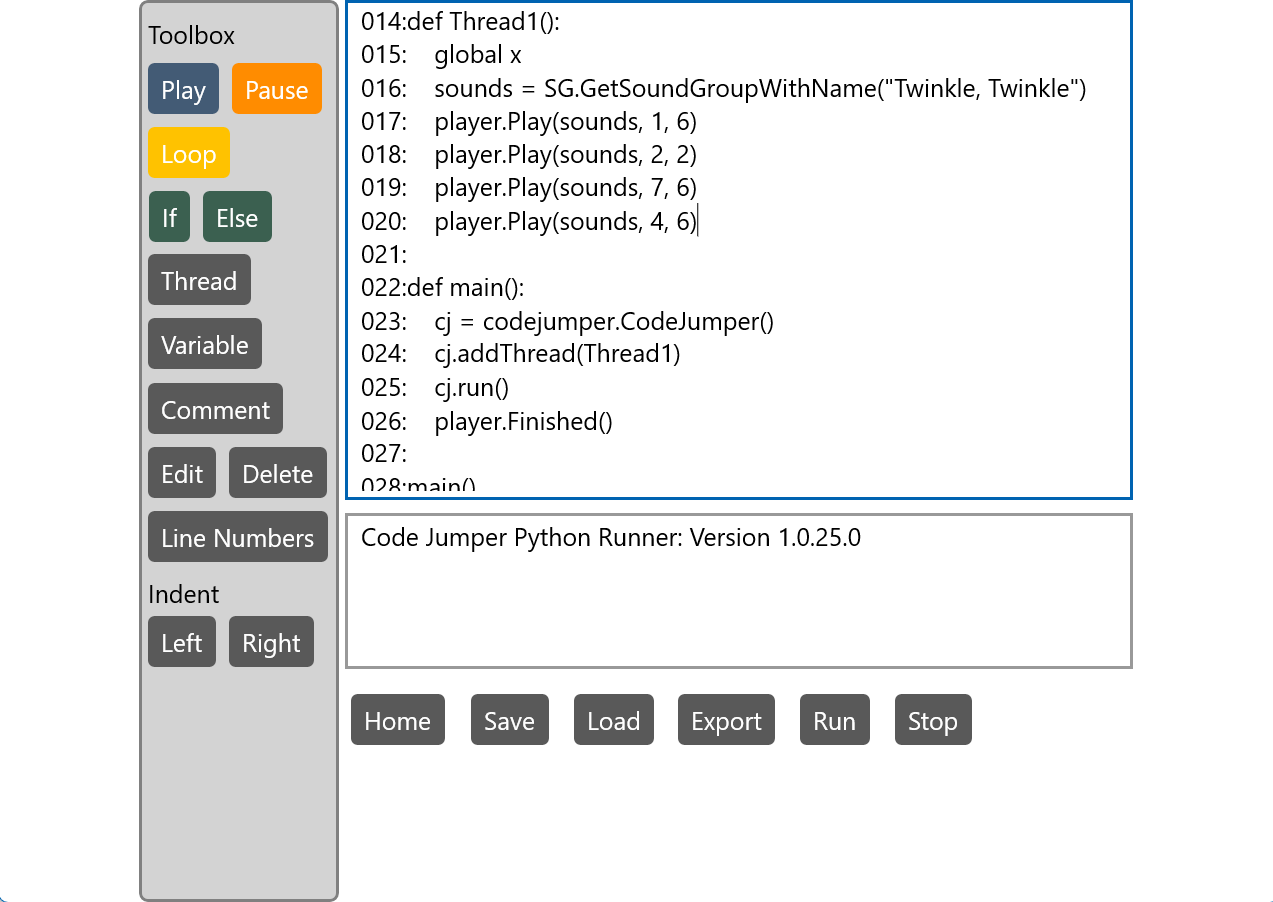


## Explore the layout of CJ Threads (the Python side of the app)

This screen is made up of different areas:

* On the far left of the screen is a toolbox with some shortcut buttons built just for Code Jumper.
* On the right is the actual Python code in a code window that can be changed and edited.
* Underneath the code window is the output window, which we will go over later.
* Underneath the output window are the program buttons.
* There are two buttons that are important right now: the *Home* button and the *Line Numbers* button.
  + The *Home* button flips the screen back to the original Code Jumper block coding.
  + The *Line Numbers* button toggles on and off the line numbers on the left of the code window. Make sure the line numbers are on for right now (by default the line numbers will be displayed.). The *Line Numbers* button is now located in the Toolbox.





## Listening and Tracing the Code in Python

*Note to reader: The lines numbers used in this example and lesson are specific to this program. If the user adds spacing, comments, or additional code, the line numbers will change.*

1. Listen to Lines 001–016 of the code and then pause.

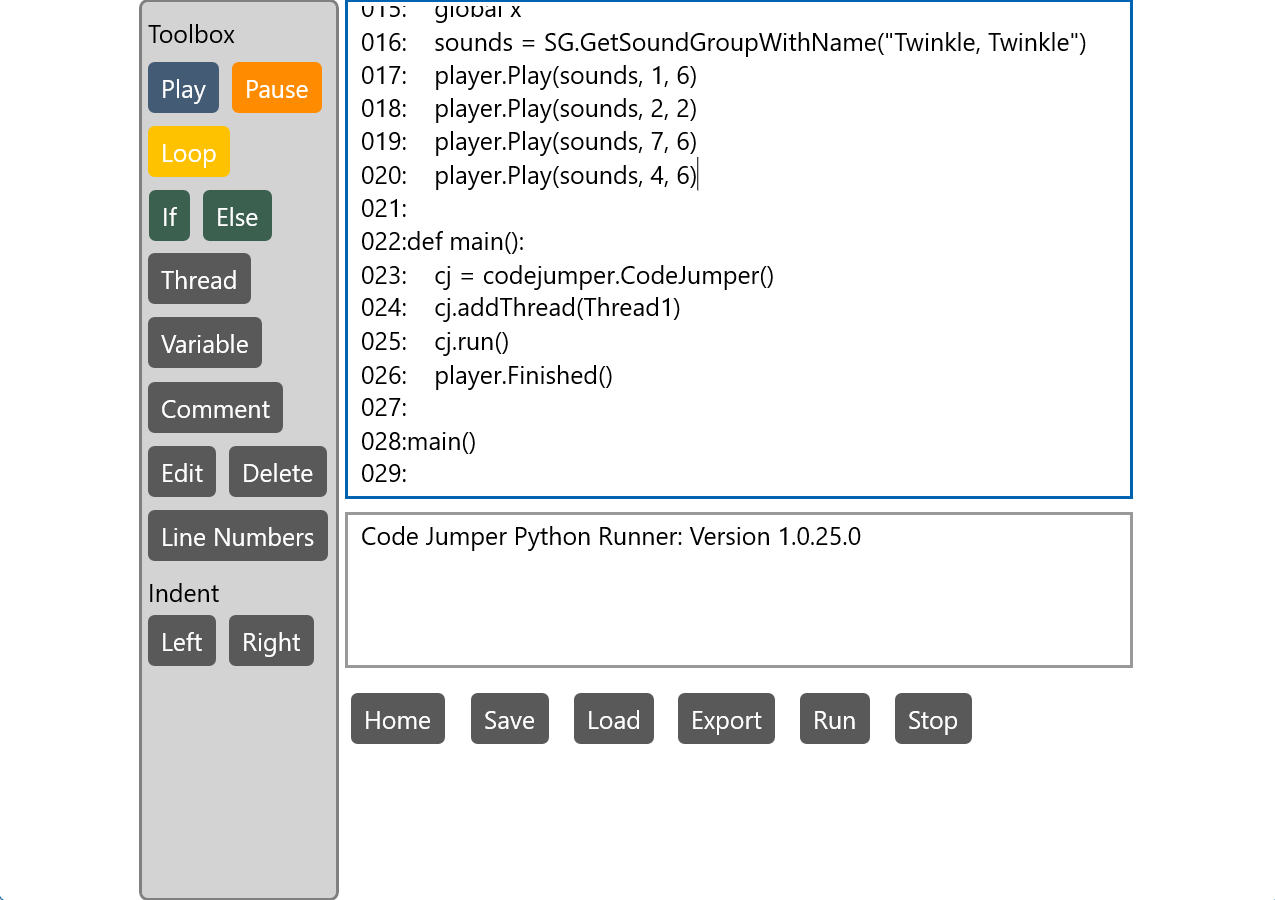
* These lines of code are important for every program and are mostly the same for all programs we are going to run, to start with.

On the block side of the app, we didn’t need to actually write the code; we made choices using the pods, dials, plugs, and buttons in the app. Those choices on the accessible block code side of the Code Jumper app told the computer details for you so you could have these options.

For example, these first lines of code tell the computer that “there is going to be a command where we can change the group of sounds they want to hear.” On the Python side, we have to actually tell the computer, or define, what we want before we can ask the computer to do anything.

1. Listen to line 011.
   * Line 11 reads: “SG = soundgroup.SoundGroups()”.
   * What do you suppose this line of code might be referring to? Does any of it sound familiar to the kinds of words that are used in Code Jumper?
   * This line is telling the computer that there’s something we need for the program (the sounds we need for the program), and we named it “soundgroup.”
   * On the block side of the app, we call it the *sound set* and we didn’t have to tell the computer there was a sound set choice—the coding was already done and it was just there to use.
   * The block side of Code Jumper sets these things for the computer but doesn’t announce all the information when we chose it. In Python, we must define things to the computer before we can use them. In Python, we are calling the sound set a *soundgroup*.
2. Listen to line 016.
   * Line 16 reads: “sounds = SG.GetSoundGroupWithName(“Twinkle, Twinkle”) What do you think this line of code is defining?
   * “Get sound group with name Twinkle, twinkle” is asking the program to go recall that thing we defined earlier, the sound group, and inside that choose the Twinkle, Twinkle sound set.
3. Listen to lines 022–029.

* These lines will be at the end of every program, just like the first 16 lines of the program.
* These last lines tell the computer that the program is finished.



1. Analogy for the first 16 lines: Think about a book. Are there parts of a book that you don’t always use? Are they still important parts of the book? Answer: Yes! The index, title page, table of contents, and copyright are all parts of the book that we assume are there and read when needed.
2. Analogy for the last 5 lines: Think about a book. Besides not having any more pages, how do we know there isn’t any more to read of the story? Answer: There are the words “The End,” there may be more pages after this, but the story is done.

## Making the Connection Between Block and Text-Based Coding

1. Refresher: Flip back to the Code Jumper side and read the code. Listen closely to what is read; listen multiple times if needed. Take notes on the following:

* What is the first thing that is read?
* What does it say to indicate a play pod?
* What is the last line that is read?

1. Flip back to CJ Threads (the Python side) and read the code. (Focus on lines 017 to 020.) Listen closely to what is read; listen multiple times and pause as needed. Take notes on the following:

* What is the first line of code read?
* What does it say to indicate a play pod? (Hint: there are four of them, so look for a line that is very similar four times in a row.)
* What is the last line that is read?

1. Are there lines of code the same on the block side (Home Screen) and Python side (CJ Threads)?
2. What is the difference between the way the code is written on the block side (Home Screen) and Python side (CJ Threads)? It may be hard to listen and spot the differences and similarities, but if you go through the line character by character, you should notice some differences. Use the table to compare and contrast the differences between block and text coding.

## How to Determine Which Sound is Playing in a Specific Line of Code in Python

The statement in the parentheses after *Play* indicates which sound plays and at which speed. The first number is the sound, and the second is the speed.

|  |  |  |
| --- | --- | --- |
| **Command** |  |  |
|  | 017: player.Play(sounds, 1, 6) | Play the sound in slot 1, which is Twinkle 1, and the speed, which is in slot 6 and plays Speed 1 |

|  |  |  |
| --- | --- | --- |
| **Sound Slot** |  | **Python** |
| Sound 1 | Twinkle 1 | (sounds, 1, x) |
| Sound 2 | Twinkle 2 | (sounds, 2, x) |
| Sound 3 | Little | (sounds, 3, x) |
| Sound 4 | Star | (sounds, 4, x) |
| Sound 5 | Twinkle 1 | (sounds, 5, x) |
| Sound 6 | Twinkle 2 | (sounds, 6, x) |
| Sound 7 | Little | (sounds, 7, x) |
| Sound 8 | Star | (sounds, 8, x) |

|  |  |  |
| --- | --- | --- |
| **Speed Slot** | **Speed** | **Python** |
| Speed 1 | 0.5 times speed | (sounds, x, 1) |
| Speed 2 | 1 times speed | (sounds, x, 2) |
| Speed 3 | 1.5 times speed | (sounds, x, 3) |
| Speed 4 | 2 times speed | (sounds, x, 4) |
| Speed 5 | 0.5 times speed | (sounds, x, 5) |
| Speed 6 | 1 times speed | (sounds, x, 6) |
| Speed 7 | 1.5 times speed | (sounds, x, 7) |
| Speed 8 | 2 times speed | (sounds, x, 8) |

\*Explanation: Sound sets have 8 slots that all need a sound in them. Since Twinkle, Twinkle only has four sounds, it repeats twice in the sound set. This also holds true for the Speed dial: There are 8 slots and four options, so they repeat. Most sound sets have 8 unique sounds. In Twinkle, Twinkle, the sound Twinkle 1 at 1 times speed can be written 4 different ways in Python:

* player.Play(sounds, 5, 6)
* player.Play(sounds, 1, 2).
* player.Play(sounds, 5, 2)
* player.Play(sounds, 1, 6)

**Note**: When a sound set has 4 sounds, all 4 sounds will repeat once. Other number of sounds in a sound set will repeat differently. For example, if there are 6 sounds, only the first 2 sounds will repeat.

# Challenge!

Play the program from the Python side and trace the code on the pods as it plays. Challenge yourself even more by reading the code and tracing the code on the pods.

# Resources

## Code Jumper and Python Code for Twinkle, Twinkle

**Thread 1 Twinkle, Twinkle:**

# My Program

import sys

import codejumper

import soundgroup

import soundplayer

import constant

from random import randrange

player = soundplayer.SoundPlayer()

player.SetSoundFolderLocation("./Sounds")

SG = soundgroup.SoundGroups()

x=1

def Thread1():

global x

sounds = SG.GetSoundGroupWithName("Twinkle, Twinkle")

**Play Twinkle 1 for 1 times speed:**

player.Play(sounds, 1, 6)

**Play Twinkle 2 for 1 times speed:**

player.Play(sounds, 2, 2)

**Play Little for 1 times speed:**

player.Play(sounds, 7, 6)

**Play Star 1 for 1 times speed:**

player.Play(sounds, 4, 6)

**End Thread:**

def main():

cj = codejumper.CodeJumper()

cj.addThread(Thread1)

cj.run()

player.Finished()

main()

# Hot Keys:

Pressing *Alt* one time will display the hot keys for the buttons (except for left and right indent).

* *Alt + P* – Play
* *Alt + U* – Pause
* *Alt + L* – Loop
* *Alt + I* – If
* *Alt + E* – Else
* *Alt + T* – Thread
* *Alt + V* – Variable
* *Alt + C* - Comment
* *Alt + D* – Edit
* *Tab* – Right Indent
* *Shift + Tab* – Left Indent
* *Delete* – Delete Line
* *Enter* – Add a Blank Line
* *Alt + H* – Home
* *Alt + N* – Line Number Toggle
* *Alt + S* – Save
* *Alt + O* – Open
* *Alt + X* – Export
* *Alt + R* – Run
* *F6 –* Switch from the Input Screen to the Output Screen