

Lab Exercises 2: Python Debugging with VScode

1. Introduction

This guide introduces how to debug a python script with Vscode on your own computer, and also cover advanced topics:

(1)Debug without parameters

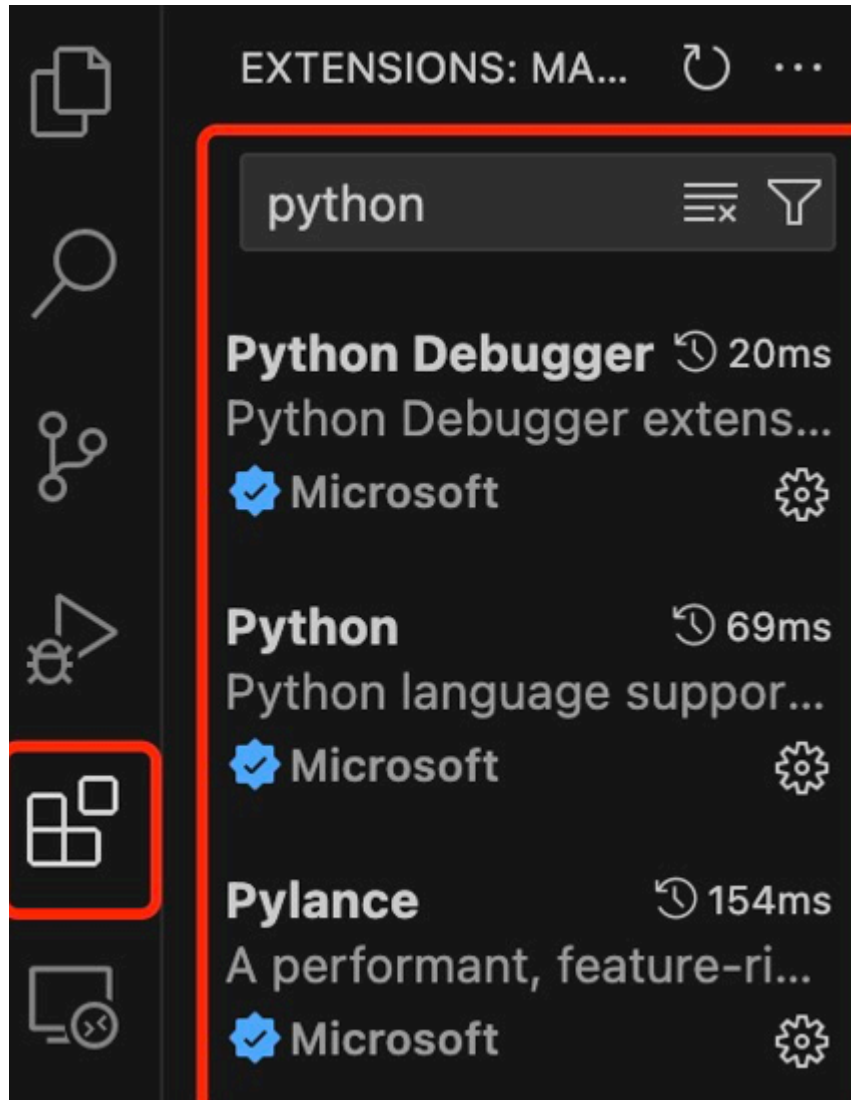
(2)Debug with parameters

2. Installation

The following is a basic installation tutorial for Python extentions. On macOS and Windows systems, the operations are similar.

2.1 Install python extention locally

First, open VSCode and search for the Python extension. Install it, and it will automatically install both the Python and Python Debugger extensions. If your network is stable, the installation should complete within a few minutes.



Similarly, search for and install the Pylance extension. This extension provides automatic code completion and syntax highlighting for your Python code, making it very useful.

2.2 Install python extention remotely (Optional)

On the server, you can follow the same steps to install the Python extension. If the external network connection is unstable, you can download the Python extension in advance from the local machine and then install it on the server. See https://code.visualstudio.com/docs/editor/extension-marketplace#_can-i-download-an-extension-directly-from-the-marketplace.

3. Debug python scripts

3.1 Debug without parameters

First, create a Python file and assume that we have already written the following code.

```
In [ ]: def add_numbers(a, b):  
        result = a + b  
        return result  
  
x = 10  
y = 20  
sum_result = add_numbers(x, y)  
print(sum_result)
```

Next, set breakpoints at any position where you want to debug. Click to the left of the line number to make a red breakpoint appear.

Then, click the **Run** button and select Python Debugger: **Debug Python File**.

Now you can check variables on the left toolbar.

On the top:

- continue
- step over
- step into
- step out
- restart
- stop

(1) Continue: Continue to next breakpoint

Continue executing the program until the next breakpoint is encountered. During program execution, the continue button will change to pause. Clicking it will stop at the current running line (for small programs, you may not see the button change to pause as it runs too fast).

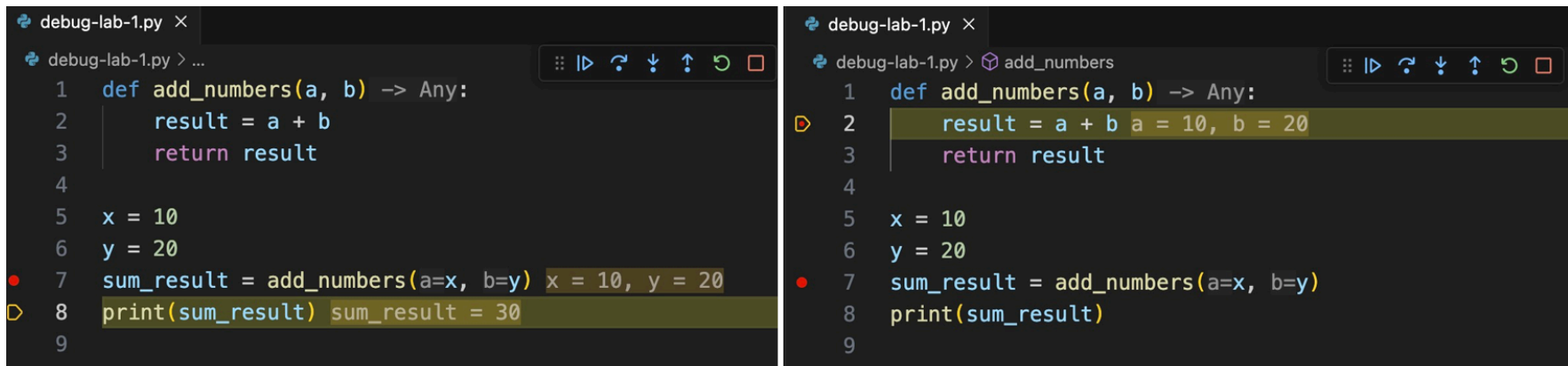
(2) Step Over: Executes the current line of code but does not go inside functions.

If the current line contains a function call, the function executes completely, and the debugger moves to the next line in the same scope.

If there is no breakpoint within the defined function, the function will be run directly without entering the function and running the code within it line by line.

Otherwise, if there is a breakpoint within the function (def block), the program will stop at the first breakpoint encountered within the custom function.

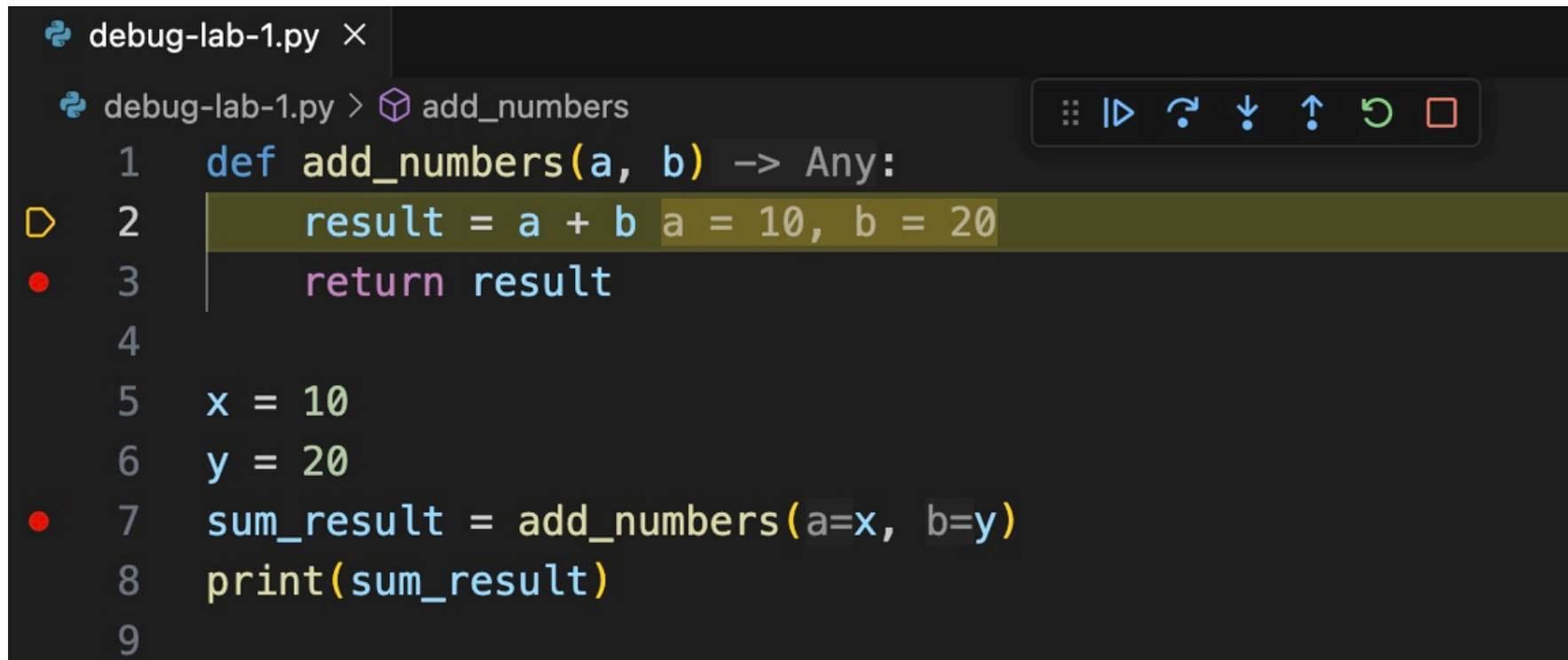
Use when you don't want to debug inside a function but just see its result.



Step Over: Customized function with breakpoints vs without breakpoints

(3) Step Into: Execute the current line and enter inside functions.

If the line contains a function, the debugger will jump inside that function regardless of whether there is a breakpoint inside the function. Therefore, you can debug it step by step. Use when you want to debug inside a function.



```
debug-lab-1.py ×
debug-lab-1.py > add_numbers
1 def add_numbers(a, b) -> Any:
2     result = a + b
3     return result
4
5 x = 10
6 y = 20
7 sum_result = add_numbers(a=x, b=y)
8 print(sum_result)
9
```

(4) Step Out

Exits the current function and returns to where it was called Useful when you accidentally Step Into a function and want to return to the main code quickly.

(5) Restart & Stop: restart your debug & stop your debug.

3.2 Debug with parameters

When debugging a Python script with arguments, we cannot directly run the debugger. First, we need to add some code to enable the Python program to accept arguments. For instance:

```
import argparse
```

```
def add_numbers(a, b):
```

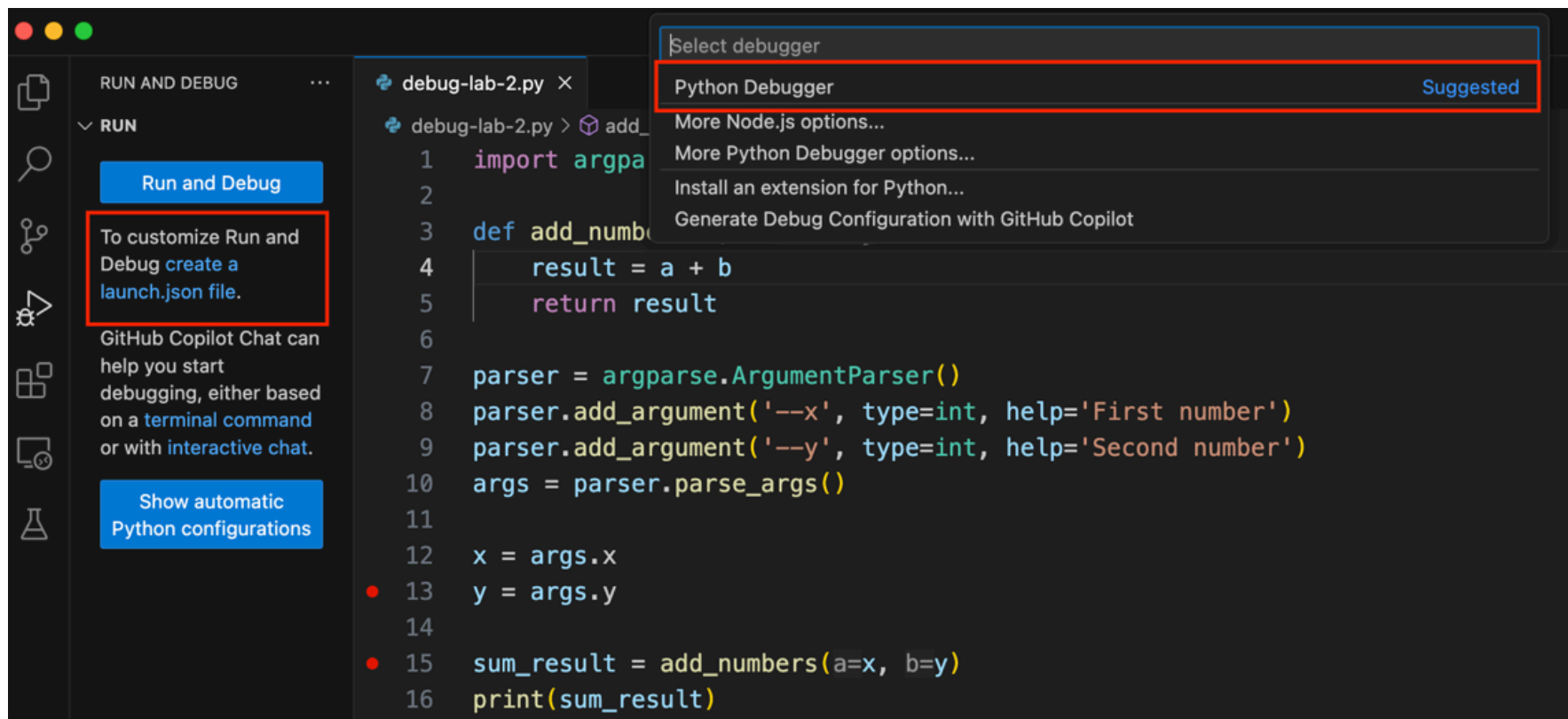
```
    result = a + b
    return result
```

```
parser = argparse.ArgumentParser()
parser.add_argument('--x', type=int, help='First number')
parser.add_argument('--y', type=int, help='Second number')
args = parser.parse_args()
```

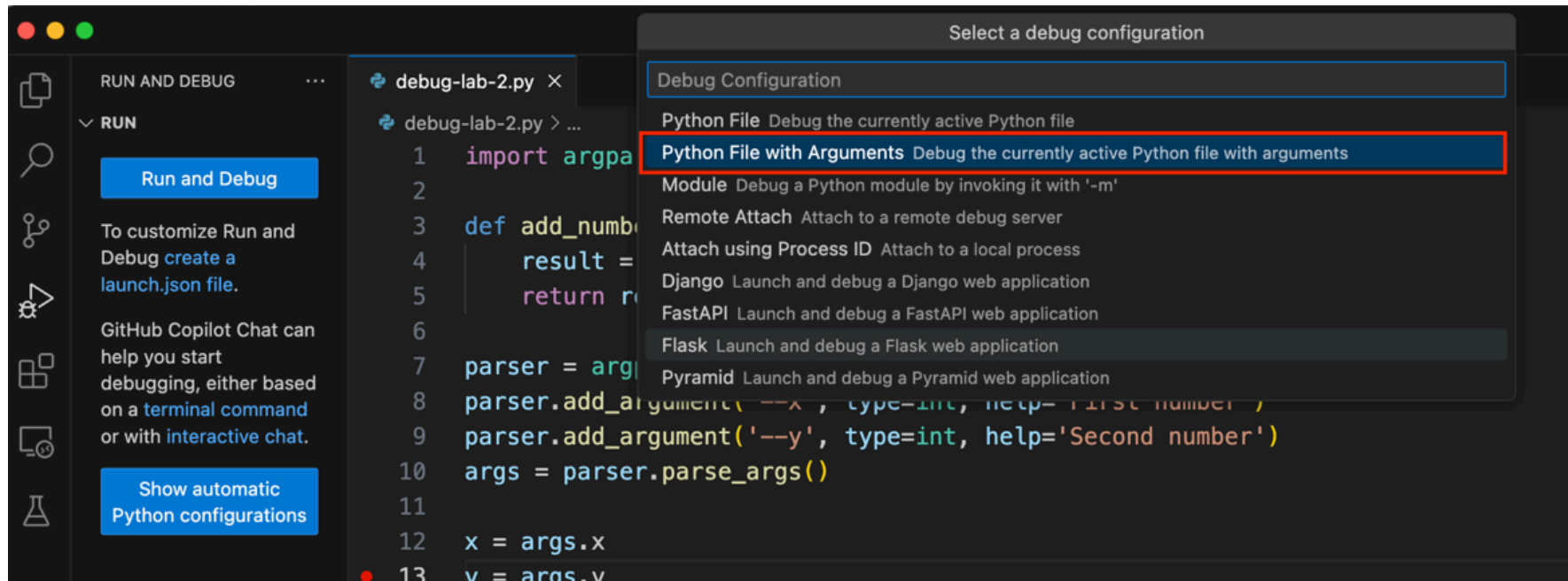
```
x = args.x
y = args.y
```

```
sum_result = add_numbers(x, y)
print(sum_result)
```

Then, we click on the debug icon and select `create a launch.json file` from the toolbar.



Next, we choose **Python Debugger -> Python File with Arguments** (Debug the currently active Python file with arguments) . VSCode will automatically generate a `launch.json` file based on our selection.



The next step is to modify the `args` parameter as shown in the image. Once the modification is complete, we return to the Python file that needs debugging and click **Python Debugger: Debug using launch.json** (Remember to create breakpoints).

debug-lab-2.py

launch.json X

.vscode > launch.json > Launch Targets > Python Debugger: Current File with Arguments

```
1  {
2      // Use IntelliSense to learn about possible attributes.
3      // Hover to view descriptions of existing attributes.
4      // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
5      "version": "0.2.0",
6      "configurations": [
7          {
8              "name": "Python Debugger: Current File with Arguments",
9              "type": "debugpy",
10             "request": "launch",
11             "program": "${file}",
12             "console": "integratedTerminal",
13             "args": [
14                 "--x", "10",
15                 "--y", "20"
16             ]
17         }
18     ]
19 }
20 }
```




Now we can debug all types of python scripts. The following tutorial would help you recognize bugs and avoid them.

4. What is a bug?

A bug is an error that causes the program to generate an unexpected output that is different from the expected output or no output.

What are some of the error codes you saw so far?

- CE - Compilation error / Syntax error
- RE - Runtime error
- WA - Wrong answer / Logical error
- TLE - Time limit exceeded

This tutorial will show more about each of these errors in examples and how to resolve them.

5. Syntax error

- **Cause:**

When you write code in any programming language, you have to follow its syntax. When the syntax of Python is not followed, you get `Syntax error`

- **Common issues:**

Missing parentheses, spelling errors, incomplete structure, indentation errors

- **Detection:**

Vscode highlights the error location

- **Solution:**

Correct the highlighted syntax error. Run the code, the output will give you some clue.

```
In [1]: name = "piyush"
        if name == "piyush"
            print(name)
```

```
Cell In[1], line 2
      if name == "piyush"
          ^
SyntaxError: expected ':'
```

5.1. Python coding principles / features

- Readability and Simplicity

Its syntax is designed to be intuitive and easy to understand.

- Indentation Defines Structure

Unlike many other languages that use curly braces `{...}`, Python uses indentation to define blocks of code.

- Dynamic Typing

In Python, you don't need to declare a variable's type explicitly. Variables are created when you first assign a value.

- Comments and Documentation

Use the `#` symbol for single-line comments and triple quotes (`"""`) for multi-line documentation.

- Following Style Guidelines (PEP8)

Python has a style guide called PEP8 that provides conventions for writing clean and consistent code.

6. Runtime error

- **Cause:**

Runtime error occurs when your syntax is correct but the compiler (or interpreter in case of Python), is still not able to run the code due to an error.

- **Common issues:**

- `NameError`: Undefined variables `print(a)`
- `TypeError`: Operation applied to incorrect type of data `result = '3' + 3`
- `ValueError`: When an operation receives a parameter of the correct type but an inappropriate value `number = int('abc')`
- `IndexError`: When using indexes beyond the range of lists, tuples, strings, etc. `lst = [1, 2, 3] print(lst[5])`
- `KeyError`, `ZeroDivisionError`, `AttributeError`, etc.

- **Detection:**

Error messages during program execution

- **Solution:**

Analyzing the error message and using debugger

```
In [2]: try:
        print(a)
    except Exception as e:
        print(f'NameError:{e}')

    try:
        result = '3' + 3
    except Exception as e:
        print(f'TypeError:{e}')

    try:
        number = int('abc')
    except Exception as e:
        print(f'ValueError:{e}')

    try:
        lst = [1, 2, 3]
        print(lst[5])
    except Exception as e:
        print(f'IndexError:{e}')
```

NameError:name 'a' is not defined

TypeError:can only concatenate str (not "int") to str

ValueError:invalid literal for int() with base 10: 'abc'

IndexError:list index out of range

7. Wrong answer / Logical error

- **Cause:**

Program runs but produces incorrect output. A logical error is an error in a program that occurs when the code compiles and runs without producing any error messages, but it does not produce the expected or desired output. Instead, it performs a different computation or provides incorrect results due to a flaw in the algorithm or logic of the program. Logical errors are the hardest to find in a program.

- **Common issues:**

Logic flaws, incorrect algorithm implementation

- **Detection:**

Test cases, edge cases, manual code review

- **Solution:**

Debug step by step, use print statements, test with sample inputs

8. Time limit exceeded

- **Cause:**

Program takes too long to execute

- **Common issues:**

Inefficient algorithms, infinite loops

- **Detection:**

Program timeout message

- **Solution:**

Optimize algorithm complexity, use more efficient data structures, check for infinite loops, performance profiling

9. Exercises

Please finish the following 3 exercises (10 points in total).

- Please take screenshots to record the debugging process of all 3 exercises and submit a report in PDF.

- You will be given a week to do it, please submit the report and debugged jupyter notebook to Canvas before next lab.
- Come up to TA(s) to check your code and report if you have finished it in class.

```
In [ ]: # Time limit exceeded (3 points)
# If your program runs for more than 3 seconds, please interrupt it manually.
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            ##### Your task: Run the code, debug the variable j and update it #####
        arr[j + 1] = key
    print("Sorted Array:", arr[:5])

arr = list(range(1000, 0, -1))
insertion_sort(arr)
```

```
In [ ]: # logic (3 points)
# Username rules:
# - Must be 5 - 10 characters long.
# - Must only contain letters and numbers.
# - Must contain at least 1 digit.
# 'R2D2', "CoderGirl", "CoderGirl", "This1IsTooLong", "High5", "pyth0n"
import string
username = 'R2D2'
is_valid = False
has_digit = False

##### Your task: Add print statements as directed to help find #####
##### and fix the logic error. #####

if len(username) >= 5 and len(username) <= 10: # Check length.
    is_valid = True

for char in username: # Loop to check the characters in username.
    if char in string.digits: # Check for a digit (0-9).
        has_digit = True
    elif char not in string.ascii_letters: # Check for non-letters.
```

```

    is_valid = False
else:
    is_valid = True

if is_valid and has_digit:
    print(f'"{username}" is a valid username.')
else:
    print((f'"{username}" is invalid.'))

```

In []: *# runtime + debug with parameters (4 points)*

```

import argparse

def process_parameters(word, first_num, second_num):
    last_letter = word[-1]
    print("The last letter in '{0}' is '{1}'".format(word, last_letter))

    print(f"For {first_num} and {second_num}:")
    print("\tSum = {0}".format(first_num + second_num))
    print("\tDifference = {0}".format(first_num - second_num))
    print("\tProduct = {0}".format(first_num * second_num))
    print("\tQuotient = {0}".format(first_num / second_num))

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Debug with parameters exercise.")
    parser.add_argument("--word", type=str, required=True, help="An english word")
    parser.add_argument("--first_num", type=int, required=True, help="First whole number")
    parser.add_argument("--second_num", type=int, required=True, help="Second whole number")
    args = parser.parse_args()

    ##### parameters: word: school, first_num: 10, second_num: 0 #####
    ##### Your task: debug with above parameters and fix the error #####
    process_parameters(args.word, args.first_num, args.second_num)

```