HORRIBLE CODE ACTIVITY ANALYSIS

UNIVERSITY OF NORTH CAROLINA AT CHARLOTTE

ITSC 3155-001

FRANCO ANTUNEZ

JOE SINSKI

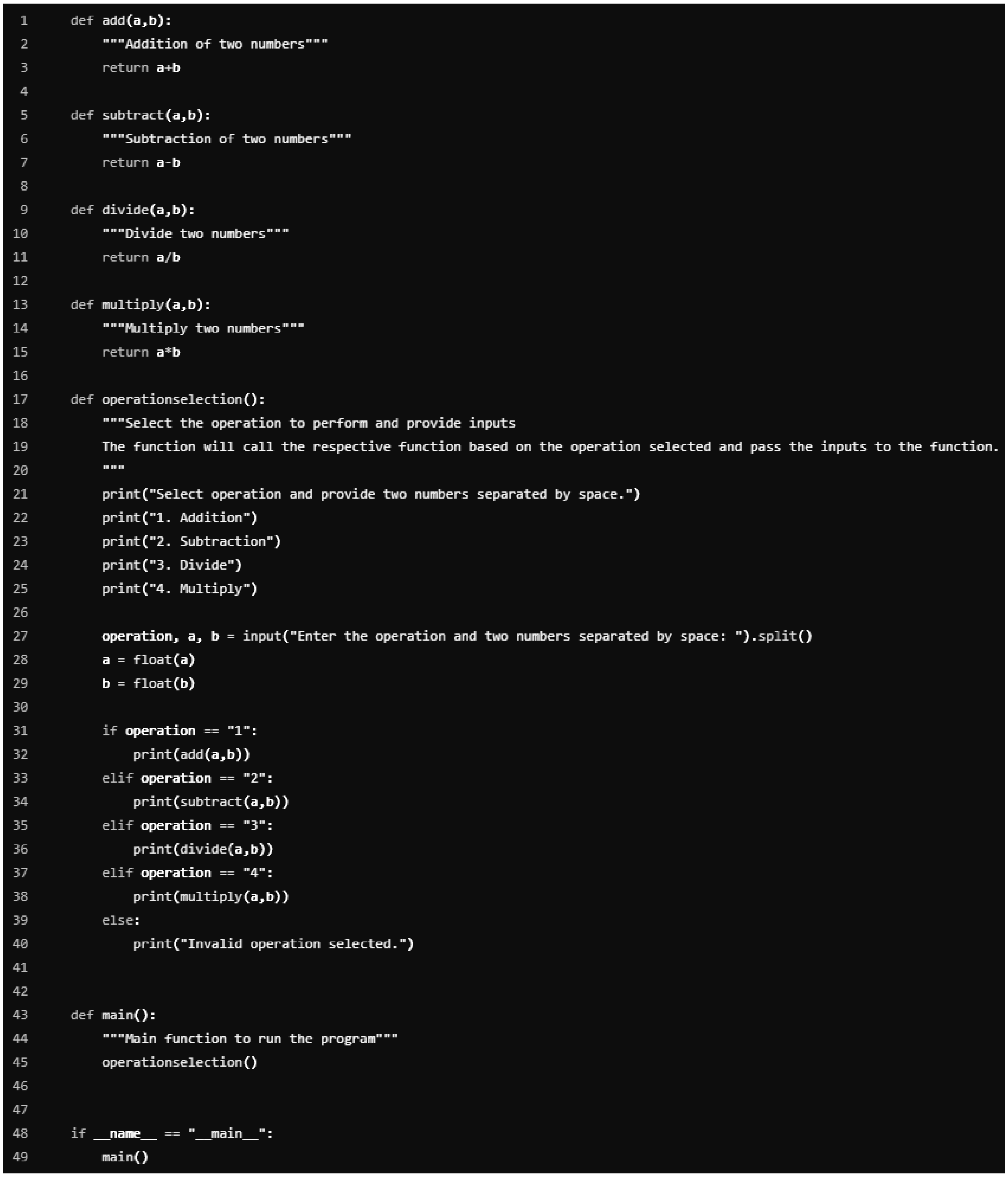
ARIEL VERA

PREFACE

A program was developed to help demonstrate coding practices that every developer should follow. The program in question is a simple calculator made with Python that only contains four arithmetic functions for fundamental math calculations, which include: adding, subtracting, multiplying, and dividing. While some may consider the program to be mediocre, the practices implemented apply to any program. For that reason, the size of the program in this example should be disregarded as we are only worried about the practices applied. Hopefully, you have the respective GitHub repository opened; however, if you do not, the documentation includes the needed code.

THE GOOD

The *calculator.py* file is an instance of a good program; the code is as follows:



```python
def add(a,b):
    """Addition of two numbers"""
    return a+b

def subtract(a,b):
    """Subtraction of two numbers"""
    return a-b

def divide(a,b):
    """Divide two numbers"""
    return a/b

def multiply(a,b):
    """Multiply two numbers"""
    return a*b

def operationselection():
    """Select the operation to perform and provide inputs
    The function will call the respective function based on the operation selected and pass the inputs to the function.
    """
    print("Select operation and provide two numbers separated by space.")
    print("1. Addition")
    print("2. Subtraction")
    print("3. Divide")
    print("4. Multiply")

    operation, a, b = input("Enter the operation and two numbers separated by space: ").split()
    a = float(a)
    b = float(b)

    if operation == "1":
        print(add(a,b))
    elif operation == "2":
        print(subtract(a,b))
    elif operation == "3":
        print(divide(a,b))
    elif operation == "4":
        print(multiply(a,b))
    else:
        print("Invalid operation selected.")


def main():
    """Main function to run the program"""
    operationselection()


if __name__ == "__main__":
    main()
```

Documentation

The program contains useful and meaningful comments that correspond to their respective functions. It takes advantage of Python docstrings and follows standard documentation standards by explaining functionalities, avoiding redundancy, and signifying segments of code.

Keep It Simple, Stupid

The source code of the program is written with simplicity mind. It is generally advised that code should be written as simple as possible. The idea is that code is written for people, not computers. The functions for the operators make good use of this rule by ensuring we only do what the function is meant to do. No operator contains an obscure algorithm that somehow correctly calculates A and B. Additionally, while we could combine the arithmetic functions into one function, that would cause the program to become bloated and because we're only working with a fixed entry of numbers, it is highly unnecessary.

You Aren't Going to Need It

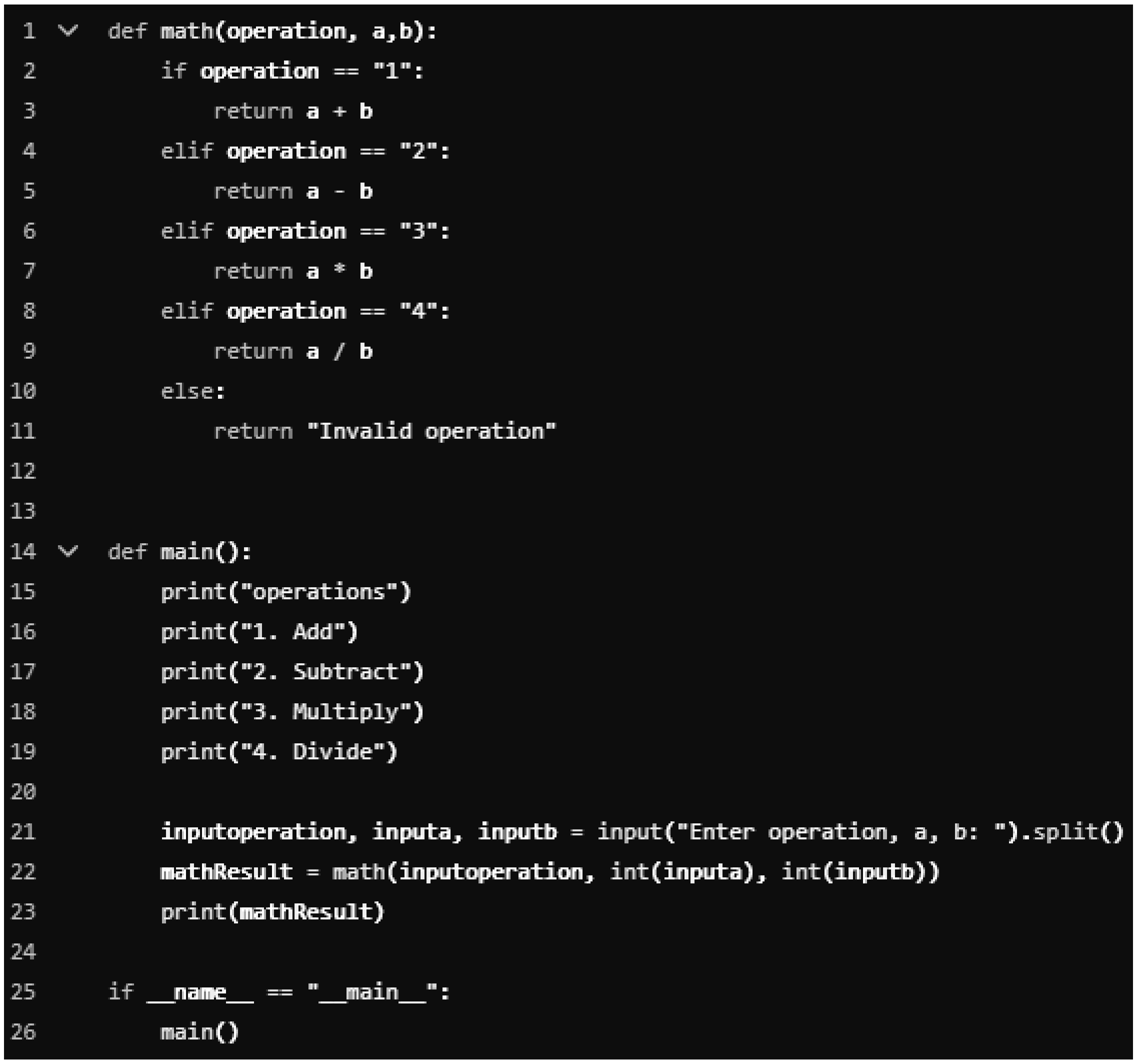
Simply put, there is not a single function within the program that is not utilized, which is great. The idea behind YAGNI is to avoid creating something that you may utilize in the future. If you do not end up using said functionality, then a solution was implemented for a non-existent problem. See the issue? You introduce unnecessary complexities that hinder the development process for yourself and others. The provided program makes use of the functions implemented, which again, is considered good practice.

Separation of Concerns

The *operationselection()* function promotes SOC by separating user interface functionality from the appropriate arithmetic functions. The arithmetic functions are intended to only perform simple equations, nothing more, nothing less. Likewise, the *operationselection()* function does so but only for user interactions. The minimization of overlapping between the functions is integral.

THE BAD

The *badCalculator.py* file is an instance of a bad program; the code is as follows:



```python
def math(operation, a,b):
    if operation == "1":
        return a + b
    elif operation == "2":
        return a - b
    elif operation == "3":
        return a * b
    elif operation == "4":
        return a / b
    else:
        return "Invalid operation"


def main():
    print("operations")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")

    inputoperation, inputa, inputb = input("Enter operation, a, b: ").split()
    mathResult = math(inputoperation, int(inputa), int(inputb))
    print(mathResult)

if __name__ == "__main__":
    main()
```

Documentation
The absence of comments in the source code is indicative of bad practice since it provides the reader with no guidance. The reader should not spend their time understanding how the code works, but rather what the code does. One could argue that providing no comments is worse than ones that are unclear. Why? Because of the cleanliness, which will be explained further later. Fortunately, the program is small enough to understand; however, imagine reviewing a codebase that had no comments.

Keep It Simple, Stupid
The *math()* function is an example of a violation of the KISS principle. The function body attempts to do every arithmetic operation in one function by amalgamating various operations using magic numbers. Again, because the program is small, we can bypass standard conventions; however, if we were to continue the development cycle of this program, updating the *math()* function can create issues as we would need to refactor the conditional statements and the corresponding operation that should be executed. The operations should have their own respective functions with reasonable parameters and not need to rely on magic numbers for determining the operation that it executes.

Separation of Concern
The inclusion of the user's selected operator existing in the *math()* function violates SOC because of overlap. While the function is already terrible as it is; i.e., determining the operation based using for-loops. There is no reason an arithmetic function should consider what input was received from the user. Intertwining the user's interaction with the math functionality only creates more room for disaster. A math function should only perform calculations based on the arguments it received. The function attempts to be clever by adding everything into one function.

Cleanliness
Provided with the reasonings of lack of documentation and compact functionality, the program exhibits code smell by additionally introducing magic numbers. The *math()* function makes use of magic numbers, which simply means that the function is arbitrary on what it executes. For instance, the number *5* could mean nothing as there is no functionality, yet. If we were to add a modulo operation that takes the number *5* as an operation, the functionality is now present. This makes it difficult to maintain and read the code base. The absence of comments only makes it considerably worse.