Assignment 5 – Debugging and Testing

Errors

When developing an application, there are three types of errors that can occur (syntax, runtime, logic).

Syntax Errors

A syntax error is triggered by syntax (code) that violates the rules of python. These errors are the easiest to discover and fix because they are typically caught by the IDE and the program will not compile with these errors.

Note: You will need to do this assignment in PyCharm or another IDE that allows you to break point

Common Python Syntax Errors

1. Code the following:
    a. Note the 2 errors
        i. Missing the ":" in the for loop
        ii. Should be using monthly_investment

Screen Capture #1 (2 points)

```
#! /usr/bin/env python3


def calculate_future_value(monthly_investment, yearly_interest, years):
    # convert yearly values to monthly values
    monthly_interest_rate = yearly_interest / 12 / 100
    months = years * 12

    # calculate future value
    future_value = 0.0
    for i in range(1, months)
        future_value += monthly_investment_amount
        monthly_interest = future_value * monthly_interest_rate
        future_value += monthly_interest

    return future_value

```

Runtime Errors

Runtime errors occur as you might guess while the application is running. Typically, a runtime error will throw an exception that stops the execution of a program. It's up to the programmer to handle these exceptions or the program will crash. A typical runtime error might be a program trying to access a resource on a network but the network is not on-line.

Logic Errors

Logic errors don't cause the program to crash but will generate unwanted results. A typical logic error might occur when trying to generate a value with the wrong algorithm.

Logic Errors

2. Code the following:
   a. Note the calculation error. Depending on the size of the algorithm, these errors can be very difficult to track down because by just looking at the code, if you did not know that the 11 stood for months in a year, the code looks correct.

Screen Capture #2 (2 points)

```python
#! /usr/bin/env python3


def calculate_future_value(monthly_investment, yearly_interest, years):
    # convert yearly to monthly values
    monthly_interest_rate = yearly_interest / 11 / 100
    months = years * 12

    # calculate future value
    future_value = 0.0
    for i in range(0, months):
        future_value += monthly_investment
        monthly_interest = future_value * monthly_interest_rate
        future_value += monthly_interest
    return future_value
```

Tracing Code

A simple way to trace code is to add statements to the code.

3. Modify the following:

Screen Capture #3 (4 points)

```python
#! /usr/bin/env python3


def calculate_future_value(monthly_investment, yearly_interest, years):
    print("Entering calculate future_value")
    # convert yearly to monthly values
    monthly_interest_rate = yearly_interest / 11 / 100
    months = years * 12

    # calculate future value
    future_value = 0.0
    for i in range(0, months):
        future_value += monthly_investment
        monthly_interest = future_value * monthly_interest_rate
        future_value += monthly_interest
        print("i =", i, "future value =", future_value)

    return future_value
```

4. On a side note, this is a great argument for using global constants
    a. It makes the code more “readable”
    b. Because of that, errors in logic are easier to detect

```
#! /usr/bin/env python3


MONTH_PER_YEAR = 12


def calculate_future_value(monthly_investment, yearly_interest, years):
    print("Entering calculate_future_value")
    # convert yearly values to monthly values
    monthly_interest_rate = yearly_interest / MONTH_PER_YEAR / 100
    months = years * MONTH_PER_YEAR

    # calculate future value
    future_value = 0.0
```

Top-Down

Top-down testing breaks the code into small blocks that can be tested.

5. Modify the calculate_future_value function as followings:
    a. Make sure you include the logic error (11) on line 6
    b. Make sure the indenting of the return is inline with the for loop

```
1  #! /usr/bin/env python3
2
3
4  def calculate_future_value(monthly_investment, yearly_interest, years):
5      # convert yearly to monthly values
6      monthly_interest_rate = yearly_interest / 11 / 100
7      months = years * 12
8
9      # calculate future value
10     future_value = 0.0
11     for i in range(0, months):
12         future_value += monthly_investment
13         monthly_interest = future_value * monthly_interest_rate
14         future_value += monthly_interest
15
16     return future_value
17
```

6. Now that we have the function in place, we can test just that function by calling it with predetermined values to guarantee the outcome is correct.

```
15
16     return future_value
17
18
19 def main():
20     future_value = calculate_future_value(100, 10, 5)
21     print("Future Value = ", future_value)
22
23
24 if __name__ == "__main__":
25     main()
26
```

7. With the predetermined values, our future value should be approximately 7808
8. Run the app to see what we get.
    a. Your value should be the same (8004.716457434104)

Screen Capture #4 (4 points)

```
C:\Users\Saddleback\AppData\Local\Pro
Future Value =  8004.716457434104

Process finished with exit code 0
```

9. Since the number is incorrect, we know we have an issue with the application

Breakpoints

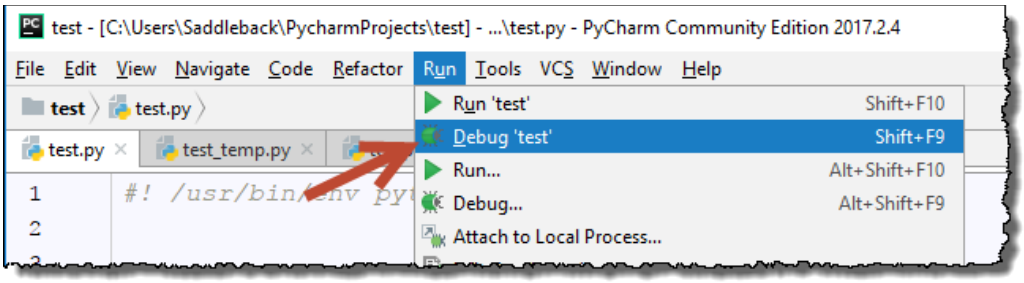
Most IDEs will include the option to set break points. In PyCharm, simply click on the blank space between the line number and code. This will add a red dot on the line and highlight the line to show there is a breakpoint.

1. Modify the calculate_future_value function as followings:

Screen Capture #5 (2 points)

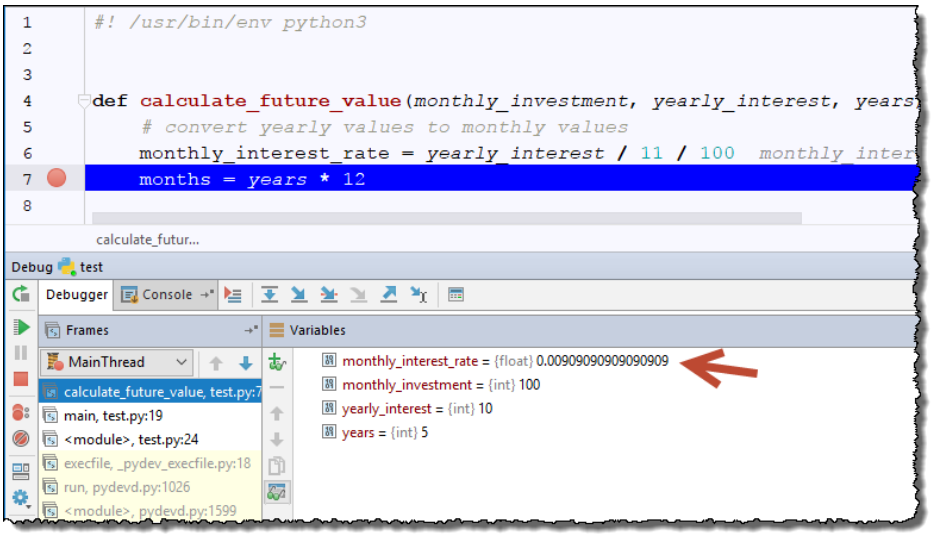
```
1   #! /usr/bin/env python3
2
3
4   def calculate_future_value(monthly_investment, yearly_interest, years):
5       # convert yearly values to monthly values
6       monthly_interest_rate = yearly_interest / 11 / 100
7       months = years * 12
8
9       # calculate future value
10      future_value = 0.0
```

2. To, make sure you run the Debug option



3. When the flow gets to the line of code, execution of the program will pause. Additionally, PyCharm (as well as most other IDEs will display value status for global and local variables)
    a. In this example, a simple manual calculation of monthly_interest_rate (monthly_interest {10} / months {12}) will show we have an error with this calculation.

Screen Capture #6 (3 points)

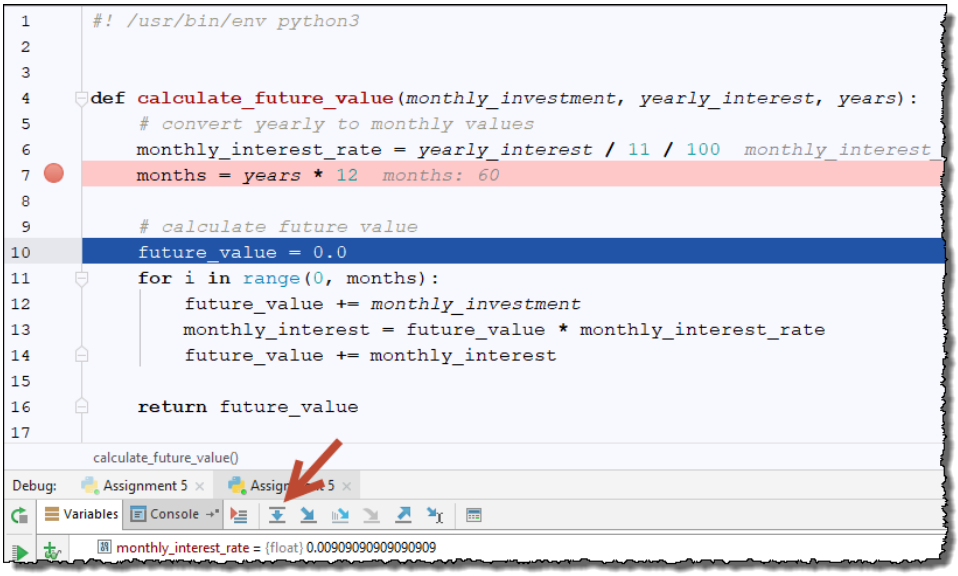


Stepping through Code

Again, most IDEs will allow you to step through the code once execution has stopped at a breakpoint.

4. Run the app again in Debug mode if needed.
5. Click on the “Step Into” or press F7 (for PyCharm) to execute the next line.

Screen Capture #7 (3points)



Extra Credit

To get full points for each extra credit, you must include screen captures of the running output as well as the python (.py) code files.

Extra Credit #1 – Tax Calculator - Debug (+1 Extra Credit)

Debug an existing program.

```
Sales Tax Calculator

Total amount: 99.99
Total after tax: 105.99
```

Specifications:

- Provided below is source code.
- Your job is to create an application using the following code, test this program, and find and fix all of the syntax, runtime, and logic errors that it contains.
- The sales tax should be 6% of the total.

```python
#!/usr/bin/env python3

TAX = 0.06

def sales_tax(total):
    sales_tax = total * tax
    return total

def main():
    print("Sales Tax Calculator\n")
    total = float(input("Enter total: "))
    total_after_tax = round(total + sales_tax(total), 2)
    print("Total after tax: ", total_after_tax)

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

Extra Credit #2 – Guessing Game - Debug (+1 Extra Credit)

Debug an existing program.

```
Guess the number!

Enter the upper limit for the range of numbers: 100
I'm thinking of a number from 1 to 100.

Your guess: 50
Too low.
Your guess: 75
Too low.
Your guess: 87
Too low.
Your guess: 94
Too low.
Your guess: 97
Too high.
Your guess: 95
Too low.
Your guess: 96
You guessed it in 7 tries.

Play again? (y/n): y

Enter the upper limit for the range of numbers: 10
I'm thinking of a number from 1 to 10.

Your guess: 5
Too low.
Your guess: 7
Too low.
Your guess: 9
Too low.
Your guess: 10
You guessed it in 4 tries.

Play again? (y/n): n

Bye!
```

Specifications:

- Provided below is source code.
- Your job is to create an application using the following code, test this program. and find and fix all of the syntax, runtime, and logic errors that it contains.

```python
#!/usr/bin/env python3

import random


def display_title():
    print("Guess the number!")
    print()


def get_limit():
    limit = int(input("Enter the upper limit for the range of numbers: "))
    return limit


def play_game(limit):
    number = random.randint(1, limit)
    print("I'm thinking of a number from 1 to " + str(limit) + "\n")
    while True:
        guess = int(input("Your guess: "))
        if guess < number:
            print("Too low.")
            count += 1
        elif guess >= number:
            print("Too high.")
            count += 1
        elif guess == number:
            print("You guessed it in " + str(count) + " tries.\n")
            return


def main():
    display_title()
    again = "y"
    while again.lower() == "y":
        limit = get_limit()
        play_game()
        again = input("Play again? (y/n): ")
        print()
    print("Bye!")


# if started as the main module, call the main function
if __name__ == "__main__":
    main()
```