Clear and uncluttered mainline
The main() function provides a clear, step-by-step structure for building the mountain bike. Each input step is separated and easy to follow, making it easy for other programmers to understand or modify.

```python
def main():
    print("\nWelcome to the MTB Builder")

    frame = Frame(...)
    fork = Fork(...)
    shock = Shock(...)
    ...
```

One logical task per subroutine
Functions like input_number() and choose_from_menu() each handle a single responsibility, whether validating numbers or displaying a selection menu. This makes the code modular and testable.

```python
def input_number(prompt, allow_float=False):
    while True:
        try:
            value = input(prompt + ": ")
            return float(value) if allow_float else int(value)
        except ValueError:
            print("Invalid input.")
```

Use of stubs
Stubs like validate_component() and backup_build() were added early during development so I could test the flow before implementing the full logic.

```python
def validate_component(component):
    return True

def backup_build(bike):
    print("\n[Stub] Simulated backup: Build not yet saved to file.")
```

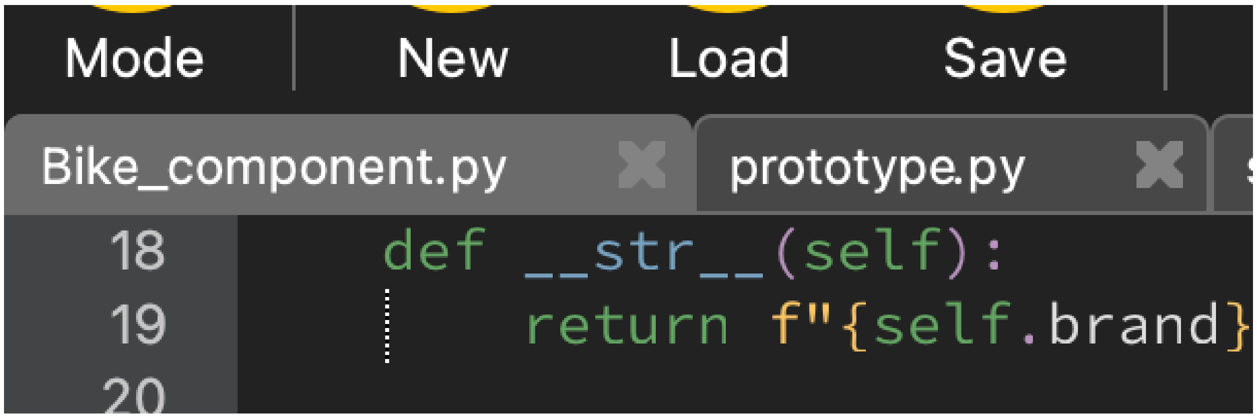
Use of control structures and data structures
I used loops (for input and validation), conditional logic (e.g. menu handling), and lists to manage multiple objects like extras. These structures kept the code efficient and scalable.

```python
extras = []
while True:
    name = input("Extra name (or 'done'): ")
    if name.lower() == 'done':
        break
    brand = input(f"{name} brand: ")
    price = input_number(f"'{name}' price", allow_float=True)
    extras.append(Extra(name, brand, price))
```

Ease of maintenance
The use of classes and inheritance (like Fork and Shock from Component) means I can update logic or attributes in one place and apply it across all components.

```python
class Component:
    def __init__(self, name, brand, price):
        self.name = name
        self.brand = brand
        self.price = float(price)
```

Version control
I used version control by first editing a file called prototype.py and then uploading the changes to my main file Bike_component.py to ensure that I always had a version of the code that works to go back to.



```
    def __str__(self):
        return f"{self.brand}
```

Regular backup

The stub backup_build() was included to simulate saving builds to file. I also made sure to save both of my files to a folder called mu_code in my icloud drive to ensure the code was saved and easily accessible.

Python

```python
def backup_build(bike):
    print("\n[Stub] Simulated backup: Build not yet saved to file.")
```