Name: Jaryd Meek

1 c++ and Errors

Complete the following two tasks for each of the following code snippets:

1. Circle the line(s) that cause an error.
2. Categorize each of the following code snippets by the type of error that they produce: runtime, compile time, or no error.
3. You may assume all needed libraries have been `#included`.

1.
```cpp
int main() {
    int a = 10;
    std::string b = "cat";
    std::cout << (a + b) << std::endl;
}
```
Compile Time

2.
```cpp
int main() {
    int a = 10;
    std::string b = "cat";
    std::cout << a << b << std::endl;
}
```
No Error

3.
```cpp
void PrintContents(std::vector<int> v) {
  for (int i = 0; i <= v.size(); i++) {
    std::cout << v[i] << std::endl;
  }
}

int main(int argc, char* argv[]) {
    std::vector<int> v = {1, 2, 4};
    PrintContents(v);
}
```
No Error

4.
```cpp
struct Book {
    std::string title;
};

void PrintContents(std::vector<Book> v) {
    for (int i = 0; i < v.size(); i++) {
        std::cout << v[i].title << std::endl;
    }
}

int main(int argc, char* argv[]) {
    Book b;
    b.title = "BFG";
    std::vector<Book> v = {b};
    PrintContents(v);
}
```
No Error

5.
```cpp
int main(int argc, char **argv) {
  std::cout << argv[0] << std::endl;
  std::cout << argv[1] << std::endl;
}
```
No Error (with 2 arguments)

Name:

2 Static type checking

1. When does static type checking happen?

At compile time

2. What are at least 3 specific benefits of static type checking?

1→ It allows errors to be found before running

2→ Typed declarations serve as automatically-checked documentation.

3→ Improves runtime efficiency

3 Python and errors

Useful tips for python:
`print(var1, var2)` is equivalent to `cout << var1 << " " << var2 << endl;`.
`range(number)` produces a list of integers from 0 to `number - 1`.
In python 3, "/" is float divide and "//" is integer divide.

1.
```python
def main():
  a = 10
  b = "cat"
  print(a + b)

main()
```
Compile time error

2.
```python
def main():
  a = 10
  b = "cat"
  print(a, b)

main()
```
No error

3.
```python
def print_list(ls):
  for i in range(len(ls) + 1):
    print(ls[i])

def main():
  ls = [1, 2, 4]
  print_list(ls)

main()
```
runtime error

4.
```python
def print_list(ls):
  for i in range(len(ls)):
    print(ls[i])

def main():
  ls = ["cat", 1236, True, False, 0.123]
  print_list(ls)

main()
```
No Error

5.
```python
import sys

def main():
  print(sys.argv[0])
  print(sys.argv[1])

main()
```
No error if 2 Arguments passed

6.
```python
def main():
  for i in range(10):
  print("Hello, world!")

main()
```
Compile time error

4 add_to_values

```
def add_to_values(ls, v):
  for i in range(len(ls)):
    ls[i] = ls[i] + v
```

1. Given the above function definition, write down 6 function calls to `add_to_values`, all with the correct number of parameters and that use a list or a string as values for the first parameter. Which of them produce errors? Make sure **at least** 2 of the function calls produce errors.

add_to_values([1,2,3], 4) No error

add_to_values([1,True,3], 4) No error

add_to_values([True,True], False) No error

add_to_values([1,2,3], True) No error

add_to_values([1,"Hello",3], 4) Error

add_to_values("Yolo", 4) Error

5 Dynamic type checking

1. When does dynamic type checking happen?

Runtime

2. What are at least 3 specific benefits of dynamic type checking?

1→ easier to work with

2→ more flexible

3→ more compact code