

SWCON104
Web & Python Programming

Advanced Features

Department of Software Convergence

Today

- Match Case
- Exception Handling
- `__name__`
- `__main__`
- `pass` Statement

Practice

- Practice_22_AdvancedFeatures
- https://www.w3schools.com/python/python_try_except.asp

Match Case

- Python 3.10 or Higher
- Structural pattern matching has been added in the form of a `match` statement and `case` statements of patterns with associated actions.
- Patterns consist of sequences, mappings, primitive data types as well as class instances.
- Pattern matching enables programs to extract information from complex data types, branch on the structure of data, and apply specific actions based on different forms of data.

Match Case: syntax and example

```
match subject:
    case <pattern_1>:
        <action_1>
    case <pattern_2>:
        <action_2>
    case <pattern_3>:
        <action_3>
    case _:
        <action_wildcard>
```

```
def number_to_string(agrument):
    match agrument:
        case 0:
            return "zero"
        case 1:
            return "one"
        case 2:
            return "two"
        case default:
            return "nothing"
```

```
print(number_to_string(0))
print(number_to_string(1))
print(number_to_string(2))
print(number_to_string(3))
print(number_to_string(4))
```

```
zero
one
two
nothing
nothing
```

Code
matchCase.py

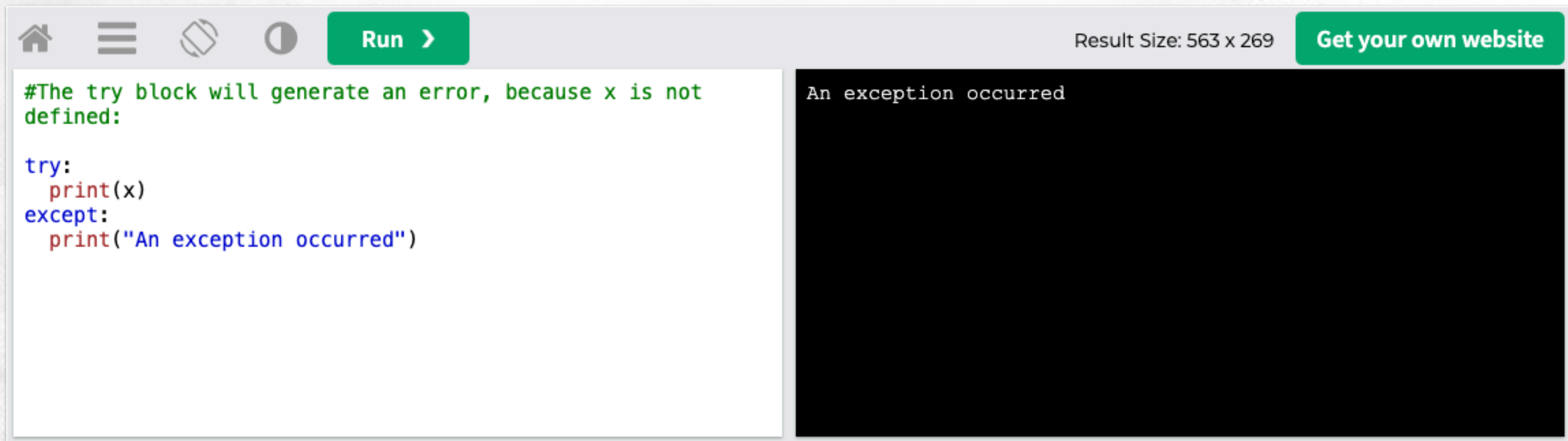
Output

Exception Handling:

- The **try** block lets you test a block of code for errors.
- The **except** block lets you handle the error.
- The **else** block lets you execute code when there is no error.
- The **finally** block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling: try..except

- When an error occurs, or exception as we call it, Python will normally stop and generate an error message.
- These exceptions can be handled using the **try** statement.
- Since the **try** block raises an error, the **except** block will be executed.
- Without the **try** block, the program will crash and raise an error.



The screenshot shows a web-based Python IDE. The top bar includes navigation icons, a green 'Run' button, and a 'Get your own website' button. The left pane contains the following Python code:

```
#The try block will generate an error, because x is not defined:

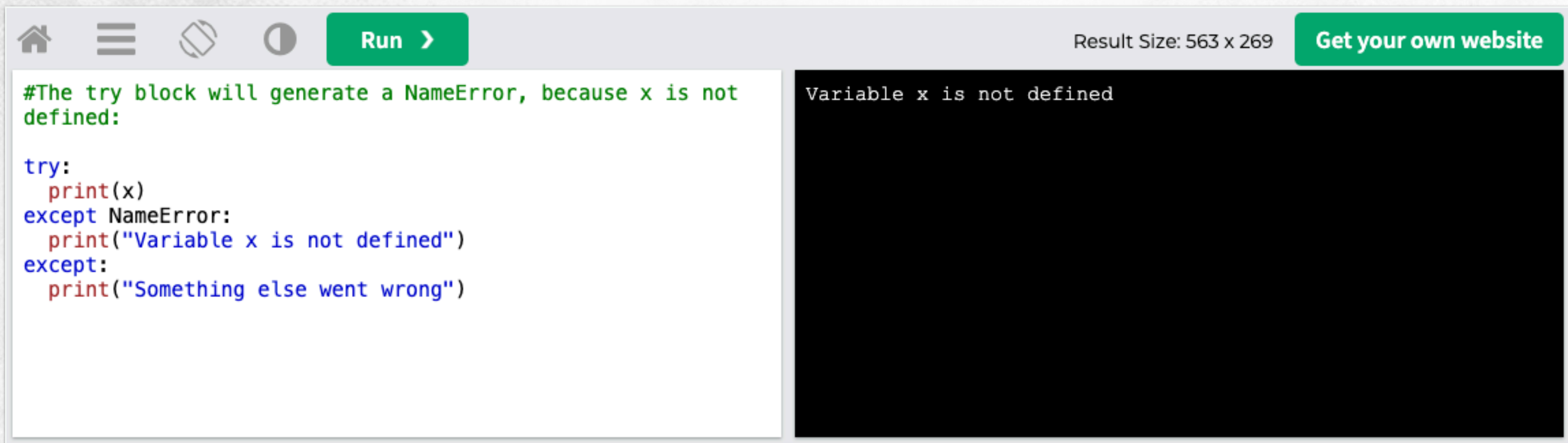
try:
    print(x)
except:
    print("An exception occurred")
```

The right pane shows the output of the code execution:

```
An exception occurred
```


Exception Handling: multiple excepts

- You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error.



The screenshot shows a web-based Python interpreter interface. At the top, there is a navigation bar with icons for home, menu, clipboard, and a toggle switch, followed by a green 'Run' button. To the right of the 'Run' button, it says 'Result Size: 563 x 269' and a green button that says 'Get your own website'. The main area is split into two panels. The left panel contains Python code: a green comment line '#The try block will generate a NameError, because x is not defined:', followed by a 'try:' block with 'print(x)', an 'except NameError:' block with 'print("Variable x is not defined")', and another 'except:' block with 'print("Something else went wrong")'. The right panel is a black terminal window showing the output 'Variable x is not defined' in white text.

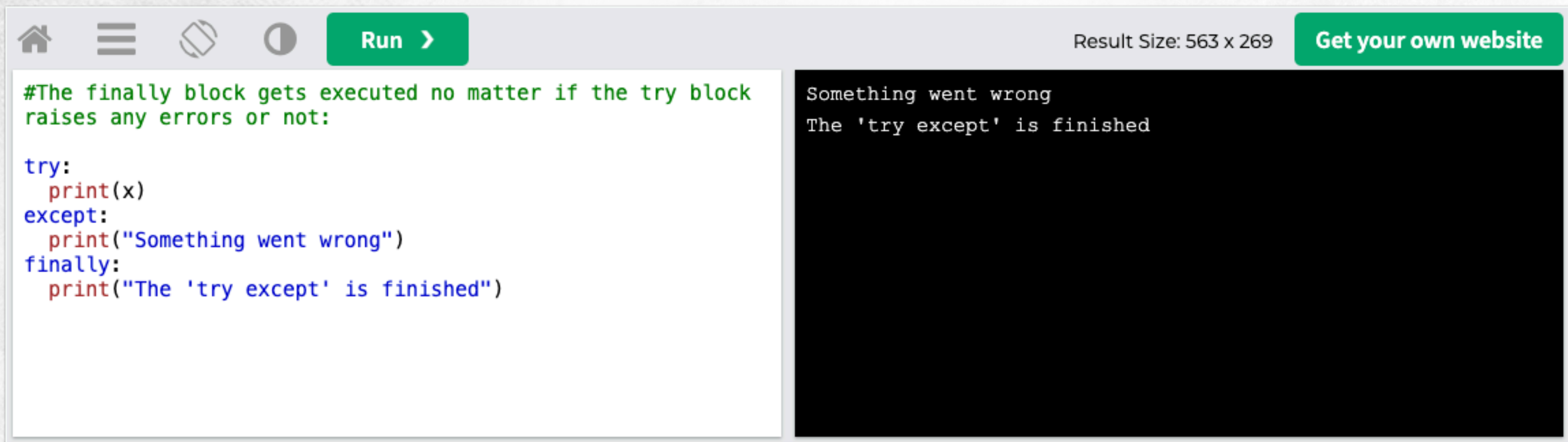
```
#The try block will generate a NameError, because x is not defined:

try:
    print(x)
except NameError:
    print("Variable x is not defined")
except:
    print("Something else went wrong")
```

Variable x is not defined

Exception Handling: finally

- The **finally** block, if specified, will be executed regardless if the **try** block raises an error or not.



The screenshot shows a web-based Python IDE. The top bar includes a home icon, a menu icon, a file icon, a play icon, a green 'Run >' button, the text 'Result Size: 563 x 269', and a green button that says 'Get your own website'. The code editor on the left contains the following Python code:

```
#The finally block gets executed no matter if the try block
raises any errors or not:

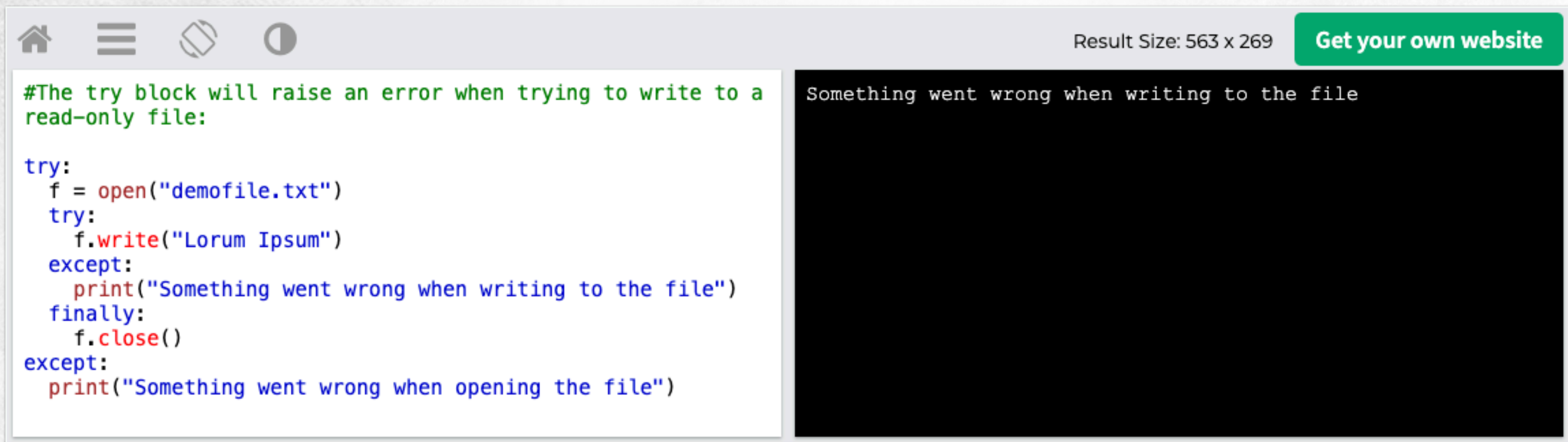
try:
    print(x)
except:
    print("Something went wrong")
finally:
    print("The 'try except' is finished")
```

The output console on the right, which has a black background, displays the following text:

```
Something went wrong
The 'try except' is finished
```


Exception Handling: finally

- **finally** can be useful to close objects and clean up resources.



The screenshot shows a web-based Python code editor interface. At the top, there are navigation icons (home, menu, copy, refresh) and a status bar indicating 'Result Size: 563 x 269' and a button 'Get your own website'. The code area on the left contains a Python script that attempts to write to a read-only file. The output area on the right shows the result of the execution.

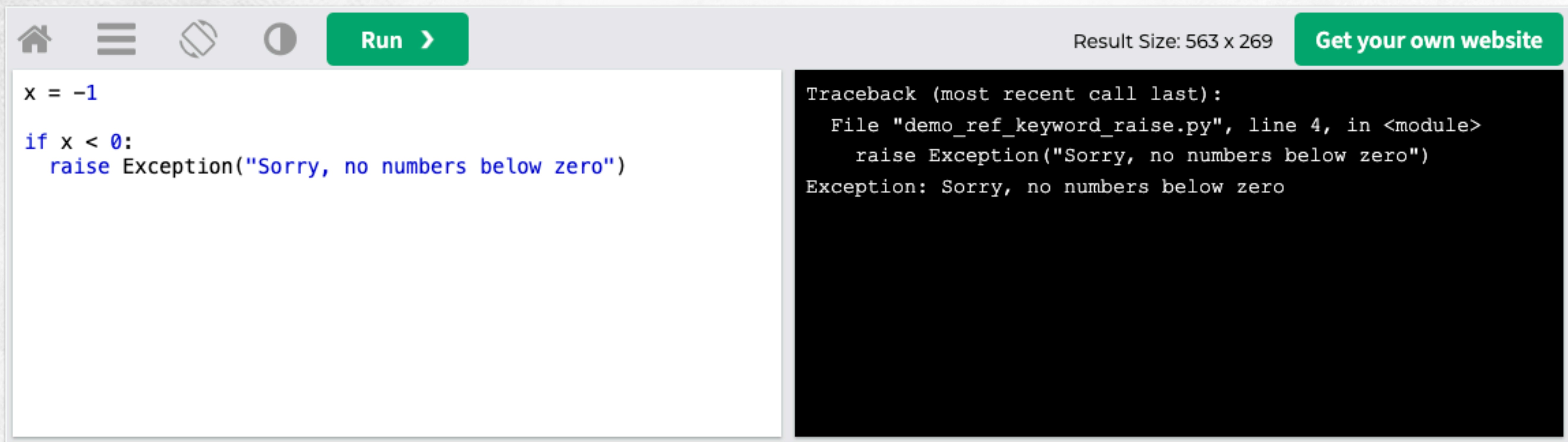
```
#The try block will raise an error when trying to write to a read-only file:

try:
    f = open("demofile.txt")
    try:
        f.write("Lorum Ipsum")
    except:
        print("Something went wrong when writing to the file")
    finally:
        f.close()
except:
    print("Something went wrong when opening the file")
```

Something went wrong when writing to the file

Exception Handling: raise an exception

- As a Python developer you can choose to throw an exception if a condition occurs.
- To throw (or raise) an exception, use the **raise** keyword.



The screenshot shows a web-based Python IDE. The top bar includes a home icon, a menu icon, a copy icon, a run icon, and a green 'Run >' button. On the right, it says 'Result Size: 563 x 269' and has a green button that says 'Get your own website'.

The code editor on the left contains the following Python code:

```
x = -1  
  
if x < 0:  
    raise Exception("Sorry, no numbers below zero")
```

The output area on the right shows the traceback for the exception:

```
Traceback (most recent call last):  
  File "demo_ref_keyword_raise.py", line 4, in <module>  
    raise Exception("Sorry, no numbers below zero")  
Exception: Sorry, no numbers below zero
```


Exception Handling: summary

try:

예외 상황이 발생 가능한 코드

except Error1:

Error1 예외 상황이 발생하는 경우 실행하는 코드

except Error2 **as** variable:

Error2 예외 상황이 발생하는 경우 variable 정보 활용하여 실행하는 코드

except:

Error1과 Error2 외의 예외 상황이 발생하는 경우 실행하는 코드

else:

예외 상황이 발생하지 않았을 때 실행하는 코드

finally:

예외 상황과 무관하게 무조건 실행하는 코드

__name__

- Within a module, the module's name (as a string) is available as the value of the global variable `__name__`.

fibonacci.py

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

```
>>> import fibo
>>> fibo.fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

Python shell

__main__

- `__main__` is the name of the environment where top-level code is run.
- “Top-level code” is the first user-specified Python module that starts running.
- It’s “top-level” because it imports all other modules that the program needs.
- Sometimes “top-level code” is called an entry point to the application.

__main__: example

fibonacci_main.py

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result

if __name__ == "__main__":
    fib(10)
```

Output

```
drsungwon~$ python fibonacci_main.py
0 1 1 2 3 5 8
```


pass statement

- The `pass` statement does nothing. It can be used when a statement is required syntactically but the program requires no action. For example:

```
>>> while True:
...     pass # Busy-wait for keyboard interrupt (Ctrl+C)
... 
```

- This is commonly used for creating minimal classes:

```
>>> class MyEmptyClass:
...     pass
... 
```


Thank you



경희대학교
KYUNG HEE UNIVERSITY