



ITS OD 303 Python



We will need to create accounts with pearsonvue if we have not already done so, we can do this with the link below.

<https://certiport.pearsonvue.com/>

It is very important that your information on this site is accurate and up-to-date, as it would be tied to your certification.

Please now take the time to create your accounts

1. Operations Using Data Types and Operators

1.1 Identify Python data types: str, int, float, bool

1.2 Data operations:

Type conversion

Indexing & slicing

Working with lists (sort, merge, append, insert, remove, find min/max, reverse)

1.3 Operator precedence: =, +=, ==, and, or, +, -, is, in, etc.

(Please excuse my dear Aunt Sally)

1.4 Select appropriate operators for desired logic



What is the data type of the result of `4 / 2` in Python 3?

- A) int
- B) float
- C) str
- D) bool

What will `my_list = [3, 1, 4]; my_list.sort(); print(my_list[-1])` output?

- A) 1
- B) 3
- C) 4
- D) Error



Which operator has the highest precedence?

- A) ==
- B) +
- C) **
- D) and

What will be the output of $5 + 3 * 2$?

- A) 16
- B) 11
- C) 21
- D) 13



3. Input and Output Operations

3.1 File I/O: open, read, write, append, file existence, with statement

3.2 Console I/O: input(), print(), formatting with .format() and f-strings, command-line args


Python's File I/O allows programs to read from and write to files using functions like open(), read(), write(), and append(). To safely handle files, the with statement is used to ensure files are closed properly. Checking file existence can be done with os.path.

Console I/O enables user interaction with input() for reading input and print() for output. Output formatting can be done using .format() or f-strings. For external input, scripts can also accept command-line arguments via sys.argv.



What will the following code print?

```
x = 10
if x > 5:
    print("A")
elif x > 8:
    print("B")
else:
    print("C")
```

- A) A
 - B) B
 - C) C
 - D) A and B
- 



What will the following code print?

```
for i in range(3):  
    if i == 1:  
        continue  
    print(i)
```

- A) 0 1 2
- B) 1 2
- C) 0 2
- D) 0 1



4. Code Documentation and Structure

4.1 Documentation: comments, docstrings, pydoc

4.2 Functions: definitions, default values, return, pass, call signatures

In Python, documentation improves code readability and includes comments (#) and docstrings (""" """) to describe functions or modules. The pydoc tool can generate documentation from these docstrings automatically.

For functions, Python uses def to define reusable code blocks. Functions can include parameters with default values, use return to send back results, or pass as a placeholder. Understanding call signatures—how functions are called with arguments—is essential for effective coding.



Which of the following opens a file for reading?

- A) open("file.txt", "w")
- B) open("file.txt", "a")
- C) open("file.txt", "r")
- D) open("file.txt", "x")

What will the following code print?

```
name = "Alice"  
print(f"Hello, {name}")
```

- A) Hello, name
- B) Hello, {name}
- C) Hello, Alice
- D) Error




What does this function return?

```
def add(x, y=3):  
    return x + y  
  
print(add(2))
```

- A) 2
- B) 3
- C) 5
- D) Error



Which of the following is a proper docstring?

- A) # This function adds numbers
 - B) "This function adds numbers"
 - C) """This function adds numbers"""
 - D) B and C
- 

5. Troubleshooting and Error Handling

5.1 Error types: syntax, logic, runtime

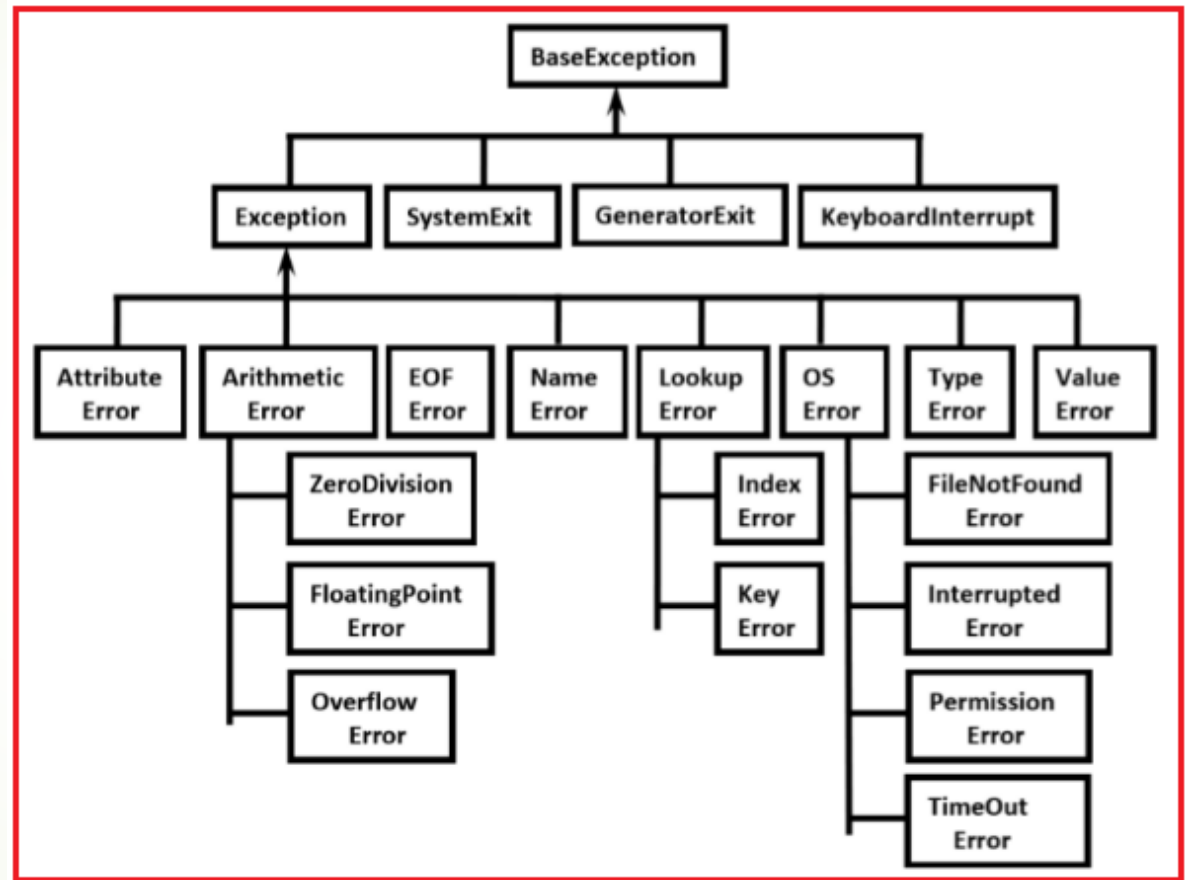
5.2 Exception handling: try, except, else, finally, raise

5.3 Unit testing: using unittest module and asserts like assertEquals, assertTrue

Python programs may encounter syntax errors (invalid code structure), logic errors (wrong output), or runtime errors (crash during execution). To handle unexpected issues, Python provides exception handling using try, except, else, finally, and raise.

For reliable code, developers also use unit testing with the unittest module, verifying behavior through assertions like assertEquals and assertTrue.

Error Types



AssertionError

```
x = "hello"
```

```
#if condition returns True, then nothing happens:
```

```
assert x == "hello"
```

```
#if condition returns False, AssertionError is raised:
```

```
assert x == "goodbye"
```

Which error will this code produce?

```
x = int("hello")
```

- A) SyntaxError
- B) TypeError
- C) ValueError
- D) NameError

What will be printed?

```
try:  
    print(1 / 0)  
except ZeroDivisionError:  
    print("Error!")
```

- A) 0
- B) 1
- C) Error!
- D) Runtime Error

6. Operations Using Modules and Tools

6.1 Built-in modules: io, os, os.path, sys

6.2 Problem-solving modules:

Math: fabs, ceil, floor, sqrt, pi, etc.

Datetime: now, strftime, weekday

Random: randint, choice, sample

Python includes many built-in modules to simplify common tasks. Modules like io, os, os.path, and sys help with file operations, directories, and command-line arguments.

For problem-solving, math provides functions like fabs, ceil, floor, sqrt, and constants like pi.

The datetime module handles dates and times with tools like now(), strftime(), and weekday().

The random module supports randomness using randint(), choice(), and sample().



Which module would you import to work with file paths?

- A) sys
- B) random
- C) os.path
- D) math

What is the output of:

```
import math  
print(math.ceil(4.2))
```

- A) 4
- B) 5
- C) 4.2
- D) Error



What is the output of:

```
import math  
print(math.ceil(4.2))
```

- A) 4
- B) 5
- C) 4.2
- D) Error



What does `random.choice([1, 2, 3])` return?

- A) Always 1
- B) An error
- C) Any random item from the list
- D) The last item in the list

What is the output of:

```
import math  
print(math.ceil(4.2))
```

- A) 4
- B) 5
- C) 4.2
- D) Error