**Syntax Errors in Python**

Syntax errors, referred to as parsing errors, are prevalent during the learning phase of Python. They occur due to improper code structure and can be easily identified by the parser, which flags the error with an arrow pointing to the earliest point of detection. The error is typically associated with the token just before the arrow.

Example:

>>> while 1 print 'Hello world'

File "<stdin>", line 1

while 1 print 'Hello world'

^SyntaxError: invalid syntax

Exceptions in Python

Even if code is syntactically correct, it can generate errors during execution known as exceptions. These errors are not always fatal and can be handled within Python programs. Most exceptions display error messages along with their type, such as ZeroDivisionError, NameError, and TypeError.

Example exceptions:

>>> 10 \* (1/0)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ZeroDivisionError: division by zero

>>> 4 + spam\*3

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'spam' is not defined

>>> '2' + 2

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: Can't convert 'int' object to str implicitly.

**Handling Exceptions**

Python provides the try statement to handle exceptions. It attempts to execute the code within the try block and if an exception occurs, the corresponding except block handles it. The except clause specifies the type of exception to catch.

Example:

>>> while True:

... try:

... x = int(input("Please enter a number: "))

... break

... except ValueError:

... print("Oops! That was no valid number. Try again...")

The try statement can have multiple except clauses for different exceptions. It allows selective handling of specific errors and can include an optional else clause to execute code when no exceptions occur.

Exception Details

Exceptions may carry associated values, and the except clause can specify a variable to capture this information. The variable is bound to the exception instance and allows access to the exception's arguments stored in instance.args.

Example:

>>> try:

... raise Exception('spam', 'eggs')

... except Exception as inst:

... print(inst.args)

... x, y = inst.args

... print('x =', x)

... print('y =', y)

Additionally, exception handlers not only deal with exceptions immediately within the try clause but also handle exceptions inside functions called within that clause.

Python's exception handling empowers developers to manage errors effectively, enabling graceful handling and recovery within programs.