In the late 1950s, computer scientists used confusing names

- **cons**: Two-argument procedure that creates a pair
- **car**: Procedure that returns the first element of a pair
- **cdr**: Procedure that returns the second element of a pair
- **nil**: The empty list

A (non-empty) list in Scheme is a pair in which the second element is `nil` or a Scheme list

- **Important!** Scheme lists are written in parentheses separated by spaces
- **A dotted list** has some value for the second element of the last pair that is not a list

```
> (cons 1 (cons 2 nil))  ; Not a well-formed list!
(1 2 3 4)
```

```
> (define x (cons 1 2))
> x
(1 . 2)
> (car x)
1
> (cdr x)
2
> (cons 1 (cons 2 (cons 3 (cons 4 nil))))
(1 2 3 4)
```

---

**Exceptions**

A built-in mechanism in a programming language to declare and respond to exceptional conditions

Python raises an exception whenever an error occurs

Exceptions can be handled by the program, preventing the interpreter from halting

Unhandled exceptions will cause Python to halt execution and print a stack trace

Mastering exceptions:

- Exceptions are objects! They have classes with constructors.
- They enable non-local continuations of control
- If `f` calls `g` and `g` calls `h`, exceptions can shift control from `h` to `f` without waiting for `g` to return.

(Exception handling tends to be slow.)
Assert Statements

Assert statements raise an exception of type AssertionError

```python
assert <expression>, <string>
```

Assertions are designed to be used liberally. They can be ignored to increase efficiency by running Python with the `-O` flag; `-O` stands for optimized

```bash
python3 -O
```

Whether assertions are enabled is governed by a bool `__debug__`

(Quiz)

Raise Statements

Exceptions are raised with a raise statement

```python
raise <expression>
```

`<expression>` must evaluate to a subclass of `BaseException` or an instance of one

Exceptions are constructed like any other object. E.g., `TypeError('Bad argument!')`

- `TypeError` -- A function was passed the wrong number/type of argument
- `NameError` -- A name wasn’t found
- `KeyError` -- A key wasn’t found in a dictionary
- `RuntimeError` -- Catch-all for troubles during interpretation

(Quiz)

Try Statements

Try statements handle exceptions

```python
try:
    <try suite>
except <exception class> as <name>:
    <except suite>
...```

Execution rule:

- The `<try suite>` is executed first
- If, during the course of executing the `<try suite>`, an exception is raised that is not handled otherwise, and
- If the class of the exception inherits from `<exception class>`, then
- The `<except suite>` is executed, with `<name>` bound to the exception

(Quiz)

Handling Exceptions

Exception handling can prevent a program from terminating

```python
>>> try:
    x = 1/0
except ZeroDivisionError as e:
    print('handling a', type(e))
    x = 0
handling a <class 'ZeroDivisionError'>
```

Multiple try statements: Control jumps to the except suite of the most recent try statement that handles that type of exception

(Quiz)

WWPD: What Would Python Display?

How will the Python interpreter respond?

```python
def invert_safe(x):
    try:
        return invert(x)
    except ZeroDivisionError as e:
        return str(e)

>>> invert_safe(0)
>>> invert_safe(1/0)
>>> invert_safe(1/0)
```

Example: Reduce

```python
def reduce(f, s, initial):
    return reduce(f, s, initial)
    # Combine elements of s pairwise using f, starting with initial.
    # E.g., reduce(mul, [2, 4, 8], 2) is equivalent to mul(mul(mul(2, 4), 8), 2).
    # >>> reduce(mul, [2, 4, 8], 2)
    # 64
    # >>>
    # f is ...
    # a two-argument function
    # s is ...
    # a sequence of values that can be the second argument
    # initial is ...
    # a value that can be the first argument
```

(Quiz)
Sierpinski's Triangle