How Are Evaluation Procedures Applied?

Evaluation rule for not expressions:
1. Evaluate the operand expression.
2. If the result is False, return True.
3. Return False.

Evaluation rule for conditional statements:
1. Evaluate the expression in the first frame of the current environment.
2. If it is a true value, execute the (if) statement.
3. If it is a false value, execute the (else) statement.

Evaluation rule for call expressions:
1. Evaluate the header expression.
2. Evaluate <exp>; The value is True if the result is a false value.
3. Simultaneously bind the arguments to the function's formal parameters.
4. Create a new function value with the specified name, argument names, and function body.
5. Create a new local frame that extends the environment with formal parameters and function body.
6. Evaluate the function body.

The most fundamental idea in computer science:
An interpreter, which determines the meaning of expressions in a programming language, is just another program.

Designing Interpreters

All programming languages are not the same!

Common elements: User-defined functions & call expressions
Some features are often excluded: Higher-order functions, object systems, while and for statements, assignment, etc.

Coming soon: The Logo language doesn’t include any of these features, but still lets us define short, powerful programs!

How can that be? Raw code is passed around like data.

Before we build interpreters:
- Recursive functions
- Recursive data structures
- Error handling

Recursive Functions

Definition: A function is called recursive if the body of that function calls itself, either directly or indirectly.

Implication: Executing the body of a recursive function may require applying that function again.

Example: Pig Latin

Yes, you’re in a college class and learning Pig Latin.

```python
def pig_latin(w):
    """Return the Pig Latin equivalent of English word w."""
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(w[1:] + w[0])

def starts_with_a_vowel(w):
    """Return whether w begins with a vowel."""
    return w[0].lower() in 'aeiou'
```

Demo
Environments for Pig Latin

```python
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(w[1:] + w[0])
```

Environments for Factorial

```python
def fact_iter(n):
    total, k = 1, 1
    while k <= n:
        total, k = total * k, k + 1
    return total

def fact(n):
    if n == 1:
        return 1
    return n * fact(n-1)
```

The Recursive Leap of Faith

```python
def fact(n):
    if n == 1:
        return 1
    return n * fact(n-1)
```

Example: Reverse a String

```python
def reverse(s):
    """Return the reverse of a string s."""
    recursive idea: The reverse of a string is the reverse of the rest of the string, followed by the first letter.
    antidisestablishmentarianism
    a ntidisestablishmentarianism
    msinairatsbnelsihtatsidit n a
```

Environments for Pig Latin

```python
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(w[1:] + w[0])
```

Environments for Factorial

```python
def fact_iter(n):
    total, k = 1, 1
    while k <= n:
        total, k = total * k, k + 1
    return total

def fact(n):
    if n == 1:
        return 1
    return n * fact(n-1)
```

The Recursive Leap of Faith

```python
def fact(n):
    if n == 1:
        return 1
    return n * fact(n-1)
```

Example: Reverse a String

```python
def reverse(s):
    """Return the reverse of a string s."""
    recursive idea: The reverse of a string is the reverse of the rest of the string, followed by the first letter.
    antidisestablishmentarianism
    a ntidisestablishmentarianism
    msinairatsbnelsihtatsidit n a
```

Base Case: The reverse of an empty string is itself.
Converting Recursion to Iteration

Hard! Iteration is a special case of recursion

Idea: Figure out what state must be maintained by the function

```python
def reverse(s):
    if s == '':
        return s
    return reverse(s[1:]) + s[0]
```

```python
def reverse_iter(s):
    r, i = '', 0
    while i < len(s):
        r, i = s[i] + r, i + 1
    return r
```

Converting Iteration to Recursion

More formulaic: Iteration is a special case of recursion

Idea: The state of an iteration can be passed as parameters

```python
def reverse_iter(s):
    r, i = '', 0
    while i < len(s):
        r, i = s[i] + r, i + 1
    return r
```

```python
def reverse2(s):
    def rev(s, r, i):
        if not i < len(s):
            return r
        return rev(s, s[i] + r, i + 1)
    return rev(s, '', 0)
```