Currying – Computing with Partially Evaluated Functions

- The idea of higher-order programming and lambda functions gives rise to the idea of partially evaluated functions.
- Again, we can look at the lambda calculus for foundations.
Currying

- Consider a lambda expression that takes a pair of values and adds them together.
- Now assume that both arguments are not immediately available...only one at a time is available
  - I know, it’s a stretch but bear with me...
- We can rewrite the lambda expression to deal with that situation by computing partially evaluated lambda expressions.
Currying

- Here is the original lambda expression expecting a pair of values,
  \[ (\lambda(x, y). x + y)(1,2) \]

- Here is a lambda expression that takes one value at a time,
  \[ (\lambda x. (\lambda y. x + y))1 \ 2 \]

- Note that after taking in the first argument it computes a partially evaluated function that expects the second argument.
Currying

Let’s take a look how the computation of the two lambda expressions differ,

\[(\lambda(x, y). x + y)(1,2) \Rightarrow x + y[(x, y) \leftarrow (1,2)] \Rightarrow x + y[x \leftarrow 1, y \leftarrow 2] \Rightarrow 1 + 2 \Rightarrow 3\]

\[(\lambda x. (\lambda y. x + y))1 2 \Rightarrow (\lambda y. x + y)[x \leftarrow 1]2 \Rightarrow (\lambda y. 1 + y)2 \Rightarrow 1 + y[y \leftarrow 2] \Rightarrow 1 + 2 \Rightarrow 3\]

Partially evaluated function
Currying

- This technique also applies to functions that take more than two values,

\[(\lambda(x, y, z). x + y + z)(1,2,3)\]

\[
\left(\lambda x. (\lambda y. (\lambda z. x + y + z))\right) 1 2 3
\]
Currying

- This technique of turning a function expecting a tuple of values to a cascade of lambda functions is called **currying**.
- It was invented by the mathematician and logician Haskell Curry.
- He developed this technique while working on combinatory logic.
Curried functions are important in the functional programming field because they make libraries for functional languages much more flexible.

We can use partially evaluated library functions to define our own functions.
Here is an example in SML taking advantage of the curried sort function.

```sml
> sml
Standard ML of New Jersey (64-bit) v110.95 [built: Sun Nov 06 00:04:31 2022]
- val sort = ListMergeSort.sort;
val sort = fn : ('a * 'a -> bool) -> 'a list -> 'a list

- (op >);
val it = fn : int * int -> bool

- (op <);
val it = fn : int * int -> bool

- sort (op <) [5, 2, 8, 3, 9, 1, 6, 7, 4];
val it = [9,8,7,6,5,4,3,2,1] : int list

- sort (op >) [5, 2, 8, 3, 9, 1, 6, 7, 4];
val it = [1,2,3,4,5,6,7,8,9] : int list

- asc_sort = sort (op >);
val asc_sort = fn : int list -> int list

- desc_sort = sort (op <);
val desc_sort = fn : int list -> int list

- asc_sort [5, 2, 8, 3, 9, 1, 6, 7, 4];
val it = [1,2,3,4,5,6,7,8,9] : int list

- desc_sort [5, 2, 8, 3, 9, 1, 6, 7, 4];
val it = [9,8,7,6,5,4,3,2,1] : int list
```
Asteroid

- Even though the modules and APIs are written in a more traditional, non-curried style in most modern programming languages, currying is still a powerful programming tool.

- Here is a simple example written in Asteroid,

```plaintext
1  -- curried function
2  function cost with tax do
3     | lambda with price do price+(price*tax/100.0)
4  end

5  -- partially evaluate function with tax rate
6  let macost = cost 6.25.
7  let ricost = cost 7.0.

10  -- show that the results are functions
11   load system type.
12   assert (type @gettype macost == "function").
13   assert (type @gettype ricost == "function").

15  -- use the functions
16   assert (macost 100.0 == 106.25).
17   assert (ricost 100.0 == 107.0).
```
Here is the same program written in Python

```python
# curried function
def cost(tax):
    return lambda price: price+(price*tax/100.0)

# partially evaluate function with tax rate
macost = cost(6.25)
ricost = cost(7.0)

# show that the results are functions
assert callable(macost)
assert callable(ricost)

# use the functions
assert (macost(100.0) == 106.25)
assert (ricost(100.0) == 107.0)
```
Currying of more than Two Arguments

- The return value is a cascade of lambda functions

```
1 function add3 with (a1,a2,a3) do
2     a1+a2+a3
3 end
4
5 assert (add3 (1,2,3) == 6).
```

```
1 function add3curr with a1 do
2     (lambda with a2 do
3         (lambda with a3 do a1+a2+a3))
4 end
5
6 assert (add3curr 1 2 3 == 6).
```
Currying Function in other Languages

- Any language that supports lambda functions and static scoping supports function currying
- This includes pretty much all languages designed over the last decade or two,
  - Python, Rust, Swift, Go, Asteroid,…