Multiple Argument Calls

With multiple arguments, just pass them in one by one, e.g.

```
(((e e1) e2) e3)
```

For example

```haskell
>>> pat 31 42 56
3038
```

1. Final = Programming Assignment except done Alone
2. O0-lambda store on CANVAS → Github used id on CANVAS
EXERCISE

Write a function `myMax` that returns the maximum of two inputs

```
myMax :: Int -> Int -> Int
myMax = ???
```

When you are done you should see the following behavior:

```
>>> myMax 10 20
20

>>> myMax 100 5
100
```
How to Return **Multiple Outputs**?

\[
\text{In} \rightarrow \text{Out} \\
(2, \text{"cat"})
\]

**Tuples**

A type for packing \( n \) different kinds of values into a single “struct”

\((T_1, \ldots, T_n)\)

For example

\[
\text{tup1 :: ???} \\
\text{tup1 = ('a', 5)}
\]

\[
\text{tup2 :: (Char, Double, Int)} \\
\text{tup2 = ('a', 5.2, 7)}
\]

**QUIZ**
What is the type of `tup3`?

\[ \text{tup3 :: ??} \]
\[ \text{tup3} = (7, 5.2, \text{True}) \]

A. (Int, Bool)

B. (Int, Double, Bool)

C. (Int, (Double, Bool))

D. (Double, Bool)

E. (Tuple, Bool)

\[ \text{(Int, Bool)} \]

**Fixed Size**

- different

---

**Extracting Values from Tuples**

We can create a tuple of three values `e1`, `e2`, and `e3` ...

\[ \text{tup} = (e1, e2, e3) \]
... but how to **extract** the values from this tuple?

**Pattern Matching**

```haskell
fst3 :: (t1, t2, t3) -> t1
fst3 (x1, x2, x3) = x1

snd3 :: (t1, t2, t3) -> t1
snd3 (x1, x2, x3) = x2

thd3 :: (t1, t2, t3) -> t1
thd3 (x1, x2, x3) = x3
```

**QUIZ**

What is the value of `quiz` defined as

```haskell
tup2 :: (Char, Double, Int)
tup2 = ('a', 5.2, 7)

quiz = snd3 tup2
```

A. 'a'

B. 5.2

C. 7

D. ('a', 5.2)
Lists

Unbounded Sequence of values of type \( T \) of any size

For example

```haskell
chars :: [Char]
chars = ['a','b','c']

ints :: [Int]
ints = [1,3,5,7]

pairs :: [(Int, Bool)]
pairs = [(1,True),(2,False)]
```
**QUIZ**

What is the type of things defined as

\[
\text{things} :: \text{???} \\
\text{things} = [ [1], [2, 3], [4, 5, 6] ]
\]

A. [Int]
B. ([Int], [Int], [Int])
C. [(Int, Int, Int)]
D. [[[Int]]] \(\checkmark\)
E. List
List’s Values Must Have The SAME Type!

The type \([T] \) denotes an unbounded sequence of values of type \(T\)

Suppose you have a list

\[
\text{oops } = [1, 2, 'c']
\]

There is no \(T\) that we can use

- As last element is not \(\text{Int}\)
- First two elements are not \(\text{Char}\)!

Result: Mysterious Type Error!
Constructing Lists

There are two ways to construct lists

```haskell
[]   -- creates an empty list
h:t  -- creates a list with "head" 'h' and "tail" t
```

For example

```haskell
>>> 3 : []
[3]

>>> 2 : (3 : [])
[2, 3]

>>> 1 : (2 : (3 : []))
[1, 2, 3]
```

Cons Operator : is Right Associative

`x1 : x2 : x3 : x4 : t` means `x1 : (x2 : (x3 : (x4 : t)))`

So we can just avoid the parentheses.

Syntactic Sugar
Haskell lets you write \([x_1, x_2, x_3, x_4]\) instead of \(x_1 : x_2 : x_3 : x_4 : []\)

### Functions Producing Lists

Let's write a function `copy3` that

- takes an input \(x\) and
- returns a list with *three* copies of \(x\)

```haskell
copy3 :: ???
copy3 x = ???
```

When you are done, you should see the following

```>>> copy3 5
[5, 5, 5]
```

```>>> copy3 "cat"
["cat", "cat", "cat"]```