1. Church Numerals

- Church numerals are a way of representing the integers in the lambda calculus.
- Church numerals are defined as functions taking two parameters:

    `0` is defined as `λf.λx. x`
    `1` is defined as `λf.λx. f x`
    `2` is defined as `λf.λx. f (f x)`
    `3` is defined as `λf.λx. f (f (f x))`
    `n` is defined as `λf.λx.` f^n `x`

- `n` has the property that for any lambda expressions `g` and `y`, `ngy` $\rightarrow^*$ $g^n y$. That is to say, `ngy` causes `g` to be applied to `y` *n* times.

2. Arithmetic

- In the lambda calculus, arithmetic functions can be represented by corresponding operations on Church numerals.
- We can define a successor function `succ` of three arguments that adds one to its first argument:

```
λn.λf.λx. f (n f x)
```

  - Example: Let us evaluate `succ 2` =

```
(λn.λf.λx. f (n f x)) (λf'.λx'. f' (f' x'))
   → λf.λx. f ((λf'.λx'. f' (f' x')) f x )
   → λf.λx. f (λx'. f (f x') x )
   → λf.λx. f (f (f x))
   = 3
```

- We can define a function `add` as follows:

```
λm.λn.λf.λx. m f (n f x)
```

  - Example: Let us evaluate `add 0 1` =

```
(λm.λn.λf.λx. m f (n f x)) 0 1
   →  λn.λf.λx. 0 f (n f x) 1
   →  λf.λx. 0 f (1 f x)
   =  λf.λx. (λf'.λx'. x') f (1 f x)
   →  λf.λx. λx'. x' (1 f x)
   →  λf.λx. (1 f x)
   =  λf.λx. ((λf'.λx'. f' x') f x)
   →  λf.λx. (λx'. f x') x
```

```
   →  λf.λx. f x
   =  1
```

- We can define a function `mult` as follows:

```
λm.λn.λf. m (n f)
```

  - Example: Let us evaluate `mul 2 3` =

```
(λm.λn.λf. m (n f )) 2 3
   →  λn.λf. 2 (n f) 3
   →  λf. 2 (3 f)
   →* λf.λx. f (f (f (f (f (f x)))))
   =  6
```

3. Logic

- The boolean value true can be represented by a function of two arguments that always selects its first argument: `λx.λy.x`
- The boolean value false can be represented by a function of two arguments that always selects its second argument: `λx.λy.y`
- An if-then-else statement can be represented by a function of three arguments `λc.λi.λe. c i e` that uses its condition `c` to select either the if-part `i` or the else-part `e`.
  - Example: Let us evaluate if true then 1 else 0:

```
(λc.λi.λe. c i e) true 1 0
   → (λi.λe. true i e) 1 0
   → (λe. true 1 e) 0
   → true 1 0
   = (λx.λy.x) 1 0
   → (λy.1) 0
   → 1
```

- The boolean operators and, or, and not can be implemented as follows:

```
and = λp.λq. p q p
or  = λp.λq. p p q
not = λp.λa.λb. p b a
```

  - Example: Let us evaluate `not true`:

```
(λp.λa.λb. p b a) true
   → λa.λb. true b a
   = λa.λb. (λx.λy.x) b a
   → λa.λb. (λy.b) a
   → λa.λb. b
   = false  (under renaming)
```

4. Other Language Constructs

- We can readily implement other programming language constructs in the lambda calculus. As an example, here are lambda calculus expressions for various list operations such as cons (constructing a list), head (selecting the first item from a list), and tail (selecting the remainder of a list after the first item):

```
cons = λh.λt.λf. f h t
head = λl.l (λh.λt. h)
tail = λl.l (λh.λt. t)
```

5. The Influence of the Lambda Calculus on Programming Languages

- The lambda calculus is the programming model for functional languages such as Haskell, ML, and OCaml.
- Constructs such as lambda expressions have appeared in many other languages.