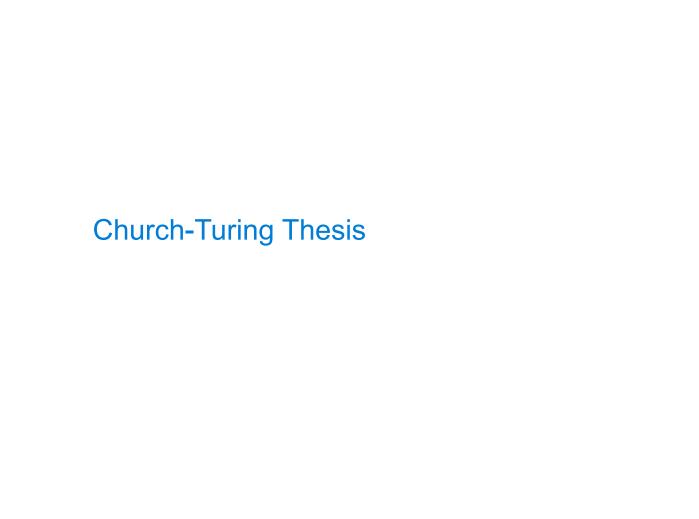


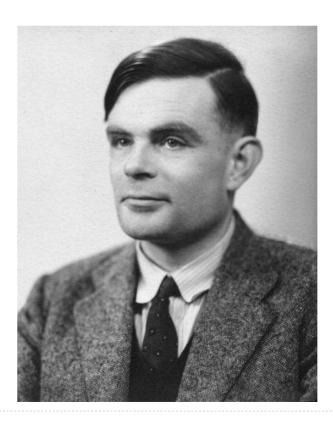
Announcements

cs61a.org/extra.html

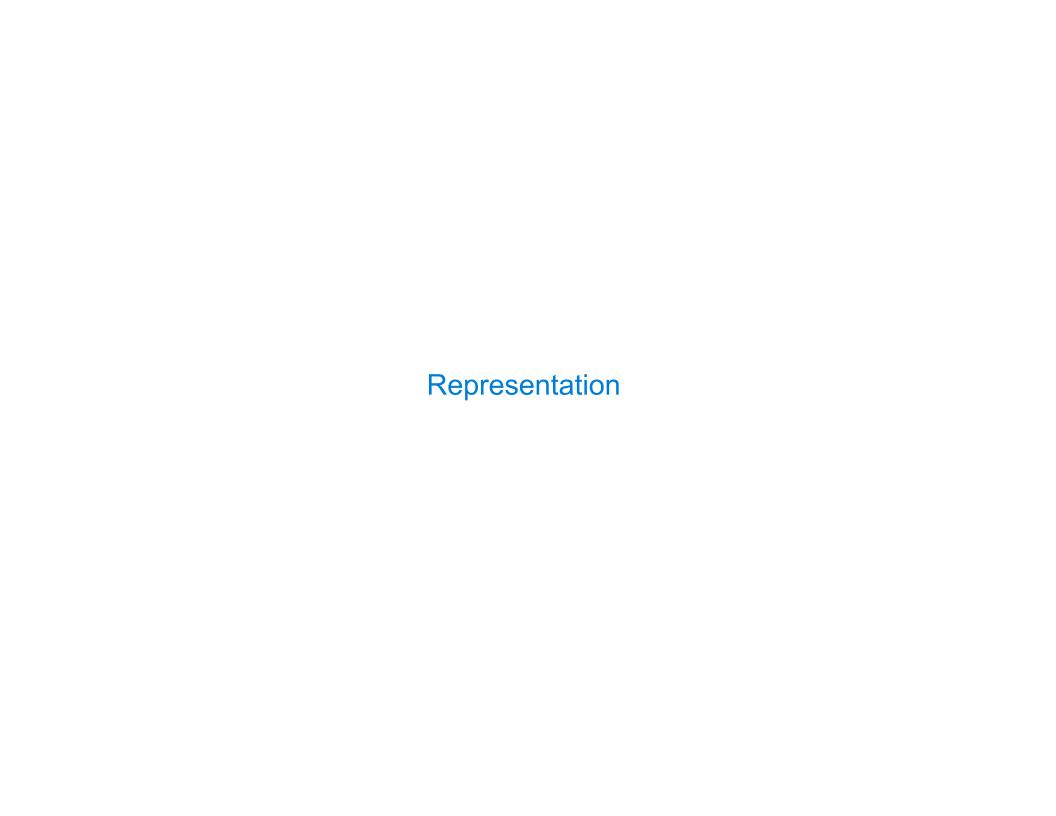


The Church-Turing Thesis

A function on the natural numbers is computable by a human following an algorithm, ignoring resource limitations, if and only if it is computable by a Turing machine.







If all we have to work with are functions and call expressions, is there any way to represent other primitive values?

```
Exercise:
t = lambda a: lambda b: a
f = lambda a: lambda b: b
                                          def f_and(p, q):
                                                                         def f_or(p, q):
def py pred(p):
                                              """Define And.
                                                                             """Define Or.
    return p(True)(False)
                                                                             >>> py_pred(f_or(t, t))
                                              >>> py_pred(f_and(t, t))
def f not(p):
                                              True
                                                                             True
    """Define Not.
                                              >>> py pred(f and(t, f))
                                                                             >>> py_pred(f_or(t, f))
                                              False
                                                                             True
    >>> py_pred(f_not(t))
                                              >>> py pred(f and(f, t))
                                                                             >>> py pred(f or(f, t))
    False
                                              False
                                                                             True
    >>> py_pred(f_not(f))
                                              >>> py pred(f and(f, f))
                                                                             >>> py pred(f or(f, f))
    True
                                                                             False
                                              False
    0.00
                                                                              .....
    return lambda a: lambda b: p(b)(a)
                                              return _____
                                                                             return _____
```

If all we have to work with are functions and call expressions, is there any way to represent other primitive values?

```
Exercise:
t = lambda a: lambda b: a
f = lambda a: lambda b: b
                                          def f_and(p, q):
                                                                          def f_or(p, q):
def py pred(p):
                                               """Define And.
                                                                              """Define Or.
    return p(True)(False)
                                               >>> py_pred(f_and(t, t))
                                                                              >>> py_pred(f_or(t, t))
def f not(p):
                                               True
                                                                              True
    """Define Not.
                                               >>> py pred(f and(t, f))
                                                                              >>> py pred(f or(t, f))
                                               False
                                                                              True
    >>> py pred(f not(t))
                                               >>> py pred(f and(f, t))
                                                                              >>> py pred(f or(f, t))
    False
                                               False
                                                                              True
    >>> py_pred(f_not(f))
                                               >>> py pred(f and(f, f))
                                                                              >>> py pred(f or(f, f))
    True
                                               False
                                                                              False
    0.00
                                                                               .....
                                              return p(q)(f)
    return lambda a: lambda b: p(b)(a)
                                                                              return _____
```

If all we have to work with are functions and call expressions, is there any way to represent other primitive values?

```
Exercise:
t = lambda a: lambda b: a
f = lambda a: lambda b: b
                                           def f_and(p, q):
                                                                           def f_or(p, q):
def py pred(p):
                                               """Define And.
                                                                               """Define Or.
    return p(True)(False)
                                               >>> py_pred(f_and(t, t))
                                                                               >>> py_pred(f_or(t, t))
def f not(p):
                                               True
                                                                               True
    """Define Not.
                                               >>> py pred(f and(t, f))
                                                                               >>> py_pred(f_or(t, f))
                                               False
                                                                               True
    >>> py_pred(f_not(t))
                                               >>> py pred(f and(f, t))
                                                                               >>> py pred(f or(f, t))
    False
                                               False
                                                                               True
    >>> py pred(f_not(f))
                                               >>> py pred(f and(f, f))
                                                                               >>> py pred(f or(f, f))
    True
                                               False
                                                                               False
    0.00
                                               return p(q)(f)
                                                                               return p(t)(q)
    return lambda a: lambda b: p(b)(a)
```

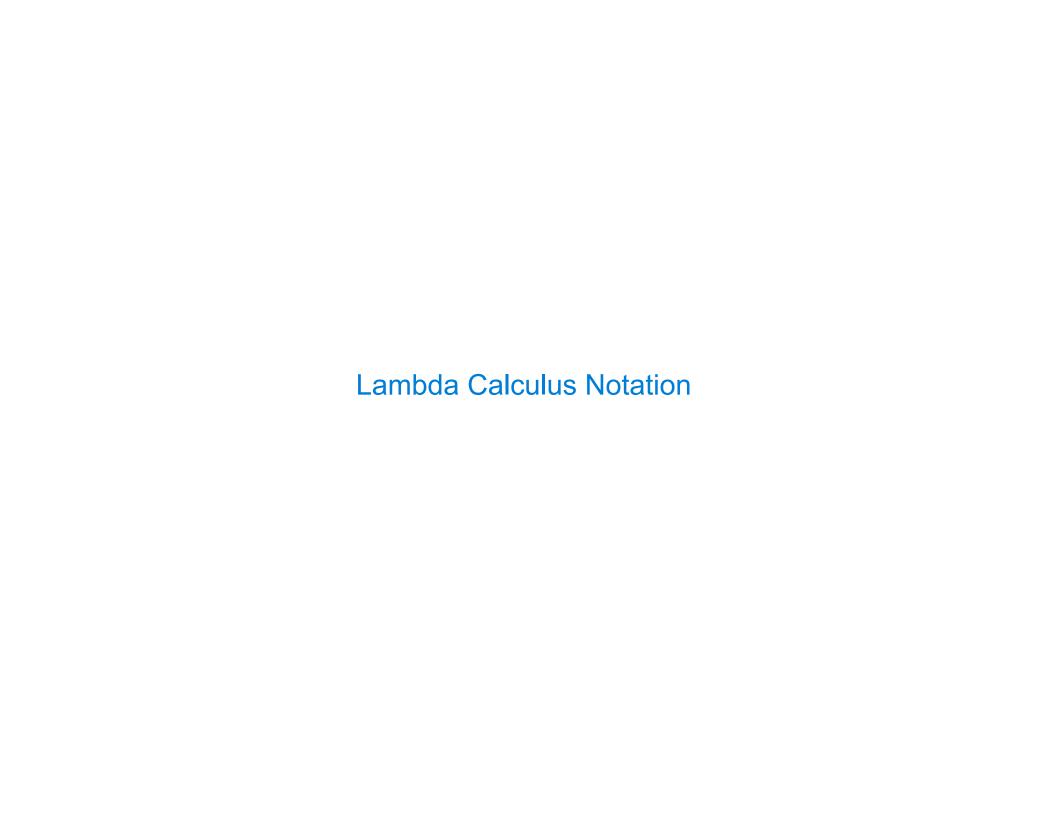
If all we have to work with are functions and call expressions, is there any way to represent other primitive values?

def f if(p, a, b):

```
"""Define If.
t = lambda a: lambda b: a
f = lambda a: lambda b: b
                                                     >>> py_pred(f_if(t, t, t))
                                                     True
def py pred(p):
                                                     >>> py_pred(f_if(t, t, f))
    return p(True)(False)
                                                     True
                                                     >>> py_pred(f_if(t, f, t))
def f not(p):
                                                     False
    """Define Not.
                                                     >>> py_pred(f_if(t, f, f))
                                                     False
    >>> py pred(f not(t))
                                                     >>> py_pred(f_if(f, t, t))
    False
    >>> py_pred(f_not(f))
                                                     >>> py_pred(f_if(f, t, f))
    True
                                                     False
                                                     >>> py_pred(f_if(f, f, t))
    return lambda a: lambda b: p(b)(a)
                                                     True
                                                     >>> py_pred(f_if(f, f, f))
                                                     False
                                                     11 11 11
```

7

return _____



Lambda Calculus

```
Variables: single letters, such as x
```

Functions: Instead of lambda x: x, write $\lambda x.x$; Instead of lambda x, y: x, write $\lambda xy.x$

Assignment: Write var f = ...

Application: Instead of f(x), write (f x); f(x)(y) and f(x, y) are both written (f x y)

Follow along! http://chenyang.co/lambda/

To type λ , just type \setminus

Boolean Values

 $var F = \lambda ab.b$

```
Variables: single letters, such as x

Functions: Instead of lambda x: x , write λx.x ; Instead of lambda x, y: x , write λxy.x

Assignment: Write var f = ...

Application: Instead of f(x) , write (f x) ; f(x)(y) and f(x, y) are both written (f x y)

Follow along! http://chenyang.co/lambda/

To type λ, just type \

var T = λab.a

Define and, or, and not!

Define exclusive or: xor(False, False) -> False
```

xor(False, True) -> True

xor(True, False) -> True
xor(True, True) -> False