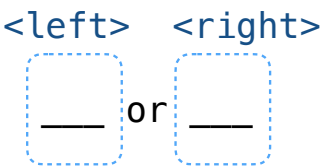
Functional Abstraction

Announcements

Conditional Expressions

Evaluating Conditional Expressions



An or expression always returns the value of one of its two sub-expressions

1. Evaluate the subexpression <left>.
2. If the result is a true value v, then the expression evaluates to v.
3. Otherwise, the expression evaluates to the value of the subexpression <right>.

```
def display(name):
    """Return their name unless it is empty,
    in which case return "Jane Doe".
    """
    return name or "Jane Doe"

def display(name, id):
    """Return their name unless it is empty,
    in which case search for their name.
    """
    return name or search_for_their_name(id)
```

Fall 2022 Midterm 1 Question 1

```
(3 and 4) - 5
```

True and False Values

The built-in bool(x) returns True for true x and False for false x.

```
>>> bool(0)
False
>>> bool(-1)
True
>>> bool(0.0)
False
>>> bool(' ')
True
>>> bool('')
False
>>> bool(False)
False
>>> bool(print('fool'))
fool
False
```

Lambda Expressions

Lambda and Def

Any program containing lambda expressions can be rewritten using def statements.

```
                          twice                          square
>>> (lambda f: lambda x: f(f(x)))(lambda y: y * y)(3)
81

>>> def twice(f):
...     def g(x):
...         return f(f(x))
...     return g
...
>>> def square(y):
...     return y * y
...
>>> twice(square)(3)
81
```

Lab 02 Q4: Composite Identity Function

Write a function that takes in two single-argument functions, `f` and `g`, and returns another **function** that has a single parameter `x`. The returned function should return whether `f(g(x))` is equal to `g(f(x))`.

(Demo)

Fall 2022 Midterm 1 Question 4(a)

(2.0 pt) Choose **all** correct implementations of `funsquare`, a function that takes a one-argument function `f`. It returns a one-argument function `f2` such that `f2(x)` has the same behavior as `f(f(x))` for all `x`.

```
>>> triple = lambda x: 3 * x
>>> funsquare(triple)(5)  # Equivalent to triple(triple(5))
45
```

A:
```
def funsquare(f):
    return f(f)
```

B:
```
def funsquare(f):
    return lambda: f(f)
```

C:
```
def funsquare(f, x):
    def g(x):
        return f(f(x))
    return g
```

D:
```
def funsquare(f):
    return lambda x: f(f(x))
```

E:
```
def funsquare(f, x):
    return f(f(x))
```

F:
```
def funsquare(f):
    def g(x):
        return f(f(x))
    return g
```

Spring 2020 Midterm 1 Question 1

```
>>> snap = lambda chat: lambda: snap(chat)
>>> snap, chat = print, snap(2020)
What is displayed here?
>>> chat()
What is displayed here?
```


https://pythontutor.com/cp/composingprograms.html#code=snap%20%3D%20lambda%20chat%3A%20lambda%3A%20snap%28chat%29%0Asnap,%20chat%20%3D%20print,%20snap%282020%29%0Achat%28%29%0A%0A&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Call Expressions

Lab 02 Q2: Higher-Order Functions

```
>>> def cake():
...     print('beets')
...     def pie():
...         print('sweets')
...         return 'cake'
...     return pie
...
>>> chocolate = cake()
beets
>>> chocolate
<function cake.<locals>.pie at ...>
>>> chocolate()
sweets
'cake'
```

```
>>> def snake(x, y):
...     if cake == more_cake:
...         return chocolate
...     else:
...         return x + y
...
>>> snake(10, 20)
<function cake.<locals>.pie at ...>
>>> snake(10, 20)()
sweets
'cake'
>>> cake = 'cake'
>>> snake(10, 20)
30
```

```
>>> more_chocolate, more_cake = chocolate(), cake
sweets
>>> more_chocolate
'cake'
```

https://pythontutor.com/cp/composingprograms.html#code=def%20cake%28%29%3A%0A%20%20%20%20print%28'beets'%29%0A%20%20%20%20def%20pie%28%29%3A%0A%20%20%20%20%20%20%20%20print%28'sweets'%29%0A%20%20%20%20%20%20%20%20return%20'cake'%0A%20%20%20%20return%20pie%0A%0Achocolate%20%3D%20cake%28%29%0A%23%20chocolate%0A%23%20chocolate%28%29%0Amore_chocolate,%20more_cake%20%3D%20chocolate%28%29,%20cake%0A%23%20more_chocolate%0Adef%20snake%28x,%20y%29%3A%0A%20%20%20%20if%20cake%20%3D%3D%20more_cake%3A%0A%20%20%20%20%20%20%20%20return%20chocolate%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20return%20x%20%2B%20y%0A%0A%23%20snake%2810,%2020%29%0Asnake%2810,%2020%29%28%29%0A%23%20cake%20%3D%20'cake'%0A%23%20snake%2810,%2020%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Assigning Names to Values

There are three ways of assigning a name to a value:

- Assignment statements (e.g., y = x) assign names in the current frame
- Def statements assign names in the current frame
- Call expressions assign names in a new local frame

```
h = lambda f: lambda x: f(f(x))
h(abs)(-3)
```

```
f = abs
x = -3
f(f(x))
```

```
h = lambda f: f(f(x))
x = -3
h(abs)
```

Currying

Function Currying

```
def make_adder(n):
    return lambda k: n + k
```

```
>>> make_adder(2)(3)
5
>>> add(2, 3)
5
```

There's a general relationship between these functions

(Demo)

Curry: Transform a multi-argument function into a single-argument, higher-order function