NESTING BLOCKS

An Introduction to Computer Science

Let's learn about Nesting Blocks.

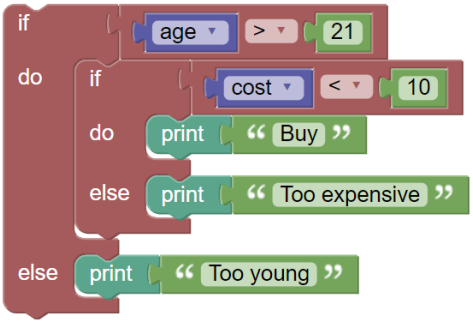
Nesting

```
if age > 21:
    if cost < 10:
        print("Buy")
    else:
        print("Too expensive")
else:
    print("Too young")
```

Nested inner IF

You can put IF statements inside of IF statements.
We call this "nesting".
As you develop more complex programs, you will do a lot of nesting.

Number of Spaces



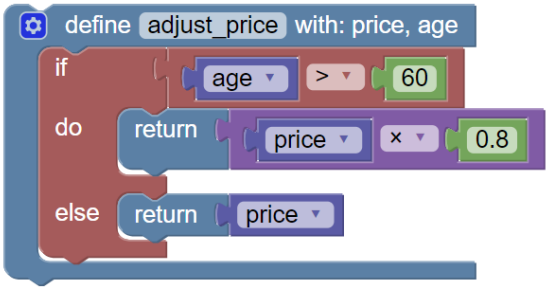

```
if age > 21:
    if cost < 10:
        print("Buy")
    else:
        print("Too expensive")
else:
    print("Too young")
```

8 Spaces total

4 Spaces

Whitespace rules can be confusing.
Every time you nest a block inside another block, the body gets indented another 4 spaces.
Observe the BlockPy blocks on the left, and their resulting whitespace on the right.

IF and Functions




```
def adjust_price(price, age):
    if age > 60:
        return price * .8
    else:
        return price
```

8 Spaces total

4 Spaces

You can put IF statements inside of functions.
In fact, this is both common and useful.
Remember the whitespace rules when this occurs.
Each nested block is another 4 spaces in.

What Goes Inside?

```
if cost > 5:
    discount = 10
price = 10
```

Third line is always executed

```
if cost > 5:
    discount = 10
    price = 10
```

Third line is SOMETIMES executed

It can be difficult to know if something belongs inside or outside of a block.
You must be aware of what you are trying to do with the IF block.
Remember: statements inside the IF block are executed only if the conditional evaluates to True.
Think critically!

ELIF block

```
if "dog" in name:
    print("Is a dog")
else:
    if "cat" in name:
        print("Is a cat")
    else:
        print("Unknown animal")
```

```
if "dog" in name:
    print("Is a dog")
elif "cat" in name:
    print("Is a cat")
else:
    print("Unknown animal")
```

ELIF keyword

In addition to the IF and ELSE blocks, there is a third type: ELIF.
The ELIF is exactly equivalent to an ELSE block with an IF inside.
However, they are sometimes used for convenience.

Two IFs vs ELSE IF

```
if "dog" in name:
    print("Is a dog")
if "cat" in name:
    print("Is a cat")
```

If name is "catdog", both execute

```
if "dog" in name:
    print("Is a dog")
elif "cat" in name:
    print("Is a cat")
```

If name is "catdog", only the first will execute

The following two pieces of code may look similar, but they are quite different.
The code on the left has two IF statements, and both will always be evaluated and potentially executed.
The code on the right has an ELIF statement, and the second will only be evaluated if the first one evaluates to false.

Unnecessary IF

```
def adjust_price(age):
    if age > 60:
        return True
    else:
        return False
```

```
def adjust_price(age):
    return age > 60
```

Two kinds of mistakes are very common with IF statements.
The first common mistake is using an IF statement when a conditional expression is fine on its own.
For example, consider this function definition that returns True if the parameter is greater than 5, or otherwise returns False.
The conditional expression already evaluates to either True or False, so it was unnecessary to use an If statement.
Instead, you can directly return the result of the conditional expression, as shown on the right.

Unnecessary Test

```
def adjust_price(age):
    return (age > 5) == True
```

```
def adjust_price(age):
    return age > 5
```

A second common mistake is to test if a Boolean expression is equal to True.
Although the expression here, "age > 5 == True", reads like it makes sense in English, it is redundant in Python.
"age > 5" already evaluates to either True or False.
If you compare a Boolean value to True, then the result is the same Boolean value.
Nothing is accomplished, you have simply made your code more complex.