If Statements in Python

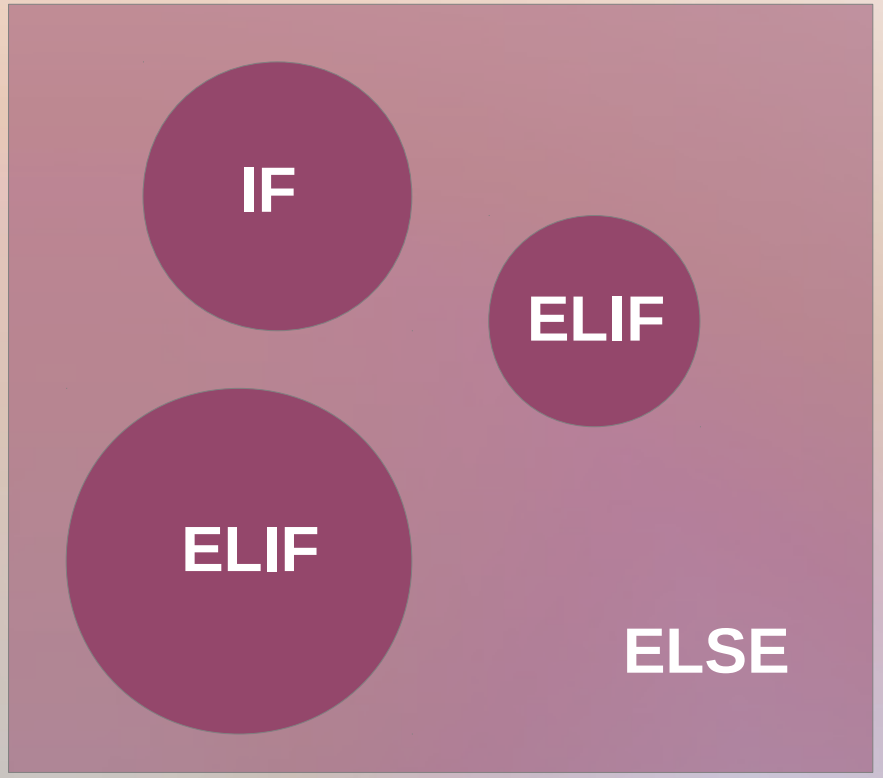
Coding choice into your programs

If statements in Python

- Previously, with dictionaries, we introduced the idea that what the code returns depends on some attribute of the input (i.e. complement['A'] = 'T')
- If statements take that to the next level:
  - You can make entire blocks of code execute only if a condition is met
  - For example, open this file only if it has 'control' in its name

First, a simple example

```
>>> mylist = ['A', 'm', 'h', 'B', '123']
>>> for m in mylist:
...     if m.isupper():
...             print 'Upper Case'
...     elif m.islower():
...             print 'Lower Case'
...     else:
...             print 'Not a letter'
...
Upper Case
Lower Case
Lower Case
Upper Case
Not a letter
```



Note that IFs and ELIFs cannot touch Each other, and anything falling outside Of these circles gets put into ELSE

Structure of an If Statement (IFS)

```
if condition:
    var.do_something()

elif another condition:
    var.do_something_else()

elif yet another condition:
    var.do_another_thing()

else:
    var.all_else_fails()
```

Optional

- All IFS's start with an “if” followed by a conditional statement and a colon
- An indent must proceed the block of code that will be executed if the condition is met
- “elif” means “else if”, meaning if the first condition is not met, try this other condition
- “else” is the final catch-all – none of the conditions specified were met, so do this

What If Statements can do

- Sorting/validating data
  - Suppose you have a list of lists with your data and you want all the lists that contain control data in one list and all the treatment data in another list
- Control for loops
  - You have a for loop running through your entire data. You want the for loop to stop running once you find something
- Pattern matching
  - You want to search DNA for a definite or fuzzy motif
- And more...

How to make a conditional statement

- A conditional statement is something that is either true or false, which we accomplish by using **operators** to compare two or more statements.
- A conditional statement with no operator just checks if that variable is not empty

```
>>> num = 30
>>> if num > 25 and num < 40:
...     print 'The number falls between 25 and 40'
...
The number falls between 25 and 40
```

Comparison operators

a = 'ATTG', b = 'ATTA', c = 'ATTG', d = 'AATTG'

a == c	(equals)
a != b	(does not equal)
a < d	(lesser than)
a <= b	(less than or equal to)
d >= b	(greater than or equal)
a in d	(a is found within d)
b not in d	(b cannot be found in d)

True or False?

Comparison operators

a = 'ATTG', b = 'ATTA', c = 'ATTG', d = 'AATTG'

a == c	(equals)
a != b	(does not equal)
a < d	(lesser than)
a <= b	(less than or equal to)
d >= b	(greater than or equal)
a in d	(a is found within d)
b not in d	(b cannot be found in d)

All statements here are TRUE

Logical operators

a = 'ATTG', b = 'ATTA', c = 'ATTG', d = 'AATTG'

- Logical operators **(and, or)** can bind comparisons together. **And** requires that both sides be true, **or** requires at least one to be true.

  a == c and a != b

  a < d and a > b

  a not in d or b not in d

  a in d and b in d

- Because they generally act like “glue” between comparisons, Python knows to evaluate comparisons first and logical operators later, like in BEDMAS

Logical operators

a = 'ATTG', b = 'ATTA', c = 'ATTG', d = 'AATTG'

- Logical operators **(and, or, not)** can bind comparisons together

  a == c and a != b (Will be true)

  a < d and a > b (Will be false)

  a not in d or b not in d (Will be true)

  a in d and b in d (Will be false)

- Because they generally act like “glue” between comparisons, Python knows to evaluate comparisons first and logical operators later, like in BEDMAS

Logical operators

a = 'ATTG', b = 'ATTA', c = 'ATTG', d = 'AATTG'

- Logical operators **(and, or, not)** can bind comparisons together

  a == c and a != b (Will be true)

  a < d and a > b (Will be false)

  a not in d or b not in d (Will be true)

  a in d and b in d (Will be false)

- Because they generally act like “glue” between comparisons, Python knows to evaluate comparisons first and logical operators later, like in BEDMAS

Sorting data

```
>>> demo
[['treatment', 'age', 'sex', 'node'],
['tumor', '45', 'F', 'pos'],
['control', '67', 'F', 'pos'],
['tumor', '32', 'M', 'pos'],
['control', '37', 'M', 'pos'],
['control', '89', 'M', 'neg'],
['tumor', '21', 'F', 'neg'],
['tumor', '52', 'M', 'neg'],
['control', '23', 'F', 'neg']]
```

```
>>> control = [ ]
>>> treatment = [ ]
>>> for line in demo:
...         if line[0] == 'control':
...                 control.append(line)
...         elif line[0] == 'tumor':
...                 treatment.append(line)
...
>>> for line in control: print line
...
['control', '67', 'F', 'pos']
['control', '37', 'M', 'pos']
['control', '89', 'M', 'neg']
['control', '23', 'F', 'neg']
>>> for line in treatment: print line
...
['tumor', '45', 'F', 'pos']
['tumor', '32', 'M', 'pos']
['tumor', '21', 'F', 'neg']
['tumor', '52', 'M', 'neg']
```

Control for loops

- There are commands you can put into if statements that can do special things to for (and while) loops
  - **break**: will stop the for loop altogether. Use this when you want to exit out of the loop early if a condition is met
  - **continue**: immediately jumps to next iteration. Use this if you know you're done with the current iteration and want to move on to the next
  - **pass**: a placeholder, does nothing

Control for loops

```
>>> demo
[['treatment', 'age', 'sex', 'node'],
['tumor', '45', 'F', 'pos'],
['control', '67', 'F', 'pos'],
['tumor', '32', 'M', 'pos'],
['control', '37', 'M', 'pos'],
['', '', '', ''],
['control', '89', 'M', 'neg'],
['tumor', '21', 'F', 'neg'],
['tumor', '52', 'M', 'neg'],
['control', '23', 'F', 'neg'],
['end', 'of', 'the', 'list'],
['', '', '', ''],
['', '', '', ''],
['', '', '', ''],
['', '', '', '']]
```

```
>>> control = []
>>> treatment = []
>>> for line in demo:
...     if line[0] == 'control':
...             control.append(line)
...     elif line[0] == 'tumor':
...             treatment.append(line)
...     elif not line[0]:
...             continue
...     elif line[0] == 'end':
...             break
...     else:
...             pass
...
```

Problems

```
>>> with open('patient_demos.csv', 'r') as file:
        data = file.read().splitlines()
```

- Open file as indicated above (get file from blog)
- Problem: Get me the ID numbers of the patients who are older than 70 and have complete fields in the tables (as in, no information is missing)
- Problem: It turns out that patients 31-66 belong to one study group, and all patients after belonged to another. You realize that you want to only study the people from the first study group. Use for loop control methods to stop the for loop once you're done with group 1.