Beginner's Python Cheat Sheet - If Statements and While Loops

What are if statements? What are while loops?

Python's if statements allow you to examine the current state of a program and respond appropriately to that state. You can write a simple if statement that checks one condition, or you can create a complex series of statements that identify the exact conditions you're interested in.

`while` loops run as long as certain conditions remain true. You can use `while` loops to let your programs run as long as your users want them to.

Conditional Tests

A conditional test is an expression that can be evaluated as true or false. Python uses the values `True` and `False` to decide whether the code in an `if` statement should be executed.

Checking for equality

A single equal sign assigns a value to a variable. A double equal sign checks whether two values are equal.

If your conditional tests aren't doing what you expect them to, make sure you're not accidentally using a single equal sign.

```
>>> car = 'bmw'
>>> car == 'bmw'
True
>>> car = 'audi'
>>> car == 'bmw'
False
```

Ignoring case when making a comparison

```
>>> car = 'Audi'
>>> car.lower() == 'audi'
True
```

Checking for inequality

```
>>> topping = 'mushrooms'
>>> topping != 'anchovies'
True
```

Numerical comparisons

Testing numerical values is similar to testing string values.

Testing equality and inequality

```
>>> age = 18
>>> age == 18
True
>>> age != 18
False
```

Comparison operators

```
>>> age = 19
>>> age < 21
True
>>> age <= 21
True
>>> age > 21
False
>>> age >= 21
False
```

Checking multiple conditions

You can check multiple conditions at the same time. The `and` operator returns `True` if all the conditions listed are true. The `or` operator returns `True` if any condition is true.

Using and to check multiple conditions

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 and age_1 >= 21
False
>>> age_1 = 23
>>> age_0 >= 21 and age_1 >= 21
True
```

Using or to check multiple conditions

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 or age_1 >= 21
True
>>> age_0 = 18
>>> age_0 >= 21 or age_1 >= 21
False
```

Boolean values

A boolean value is either `True` or `False`. Variables with boolean values are often used to keep track of certain conditions within a program.

Simple boolean values

```
game_active = True
is_valid = True
can_edit = False
```

If statements

Several kinds of if statements exist. Your choice of which to use depends on the number of conditions you need to test. You can have as many `elif` blocks as you need, and the `else` block is always optional.

Simple if statement

```
age = 19

if age >= 18:
    print("You're old enough to vote!")
```

If-else statements

```
age = 17

if age >= 18:
    print("You're old enough to vote!")
else:
    print("You can't vote yet.")
```

The if-elif-else chain

```
age = 12

if age < 4:
    price = 0
elif age < 18:
    price = 25
else:
    price = 40

print(f"Your cost is ${price}.")
```

Conditional tests with lists

You can easily test whether a certain value is in a list. You can also test whether a list is empty before trying to loop through the list.

Testing if a value is in a list

```
>>> players = ['al', 'bea', 'cyn', 'dale']
>>> 'al' in players
True
>>> 'eric' in players
False
```

Testing if two values are in a list

```
>>> 'al' in players and 'cyn' in players
```

Conditional tests with lists (cont.)

Testing if a value is not in a list

```
banned_users = ['ann', 'chad', 'dee']
user = 'erin'

if user not in banned_users:
    print("You can play!")
```

Checking if a list is empty

An empty list evaluates as `False` in an if statement.

```
players = []

if players:
    for player in players:
        print(f"Player: {player.title()}")
else:
    print("We have no players yet!")
```

Accepting input

You can allow your users to enter input using the `input()` function. All input is initially stored as a string. If you want to accept numerical input, you'll need to convert the input string value to a numerical type.

Simple input

```
name = input("What's your name? ")
print(f"Hello, {name}.")
```

Accepting numerical input using int()

```
age = input("How old are you? ")
age = int(age)

if age >= 18:
    print("\nYou can vote!")
else:
    print("\nSorry, you can't vote yet.")
```

Accepting numerical input using float()

```
tip = input("How much do you want to tip? ")
tip = float(tip)
print(f"Tipped ${tip}.")
```

While loops

A while loop repeats a block of code as long as a condition is true.

Counting to 5

```
current_number = 1

while current_number <= 5:
    print(current_number)
    current_number += 1
```

While loops (cont.)

Letting the user choose when to quit

```
prompt = "\nTell me something, and I'll "
prompt += "repeat it back to you."
prompt += "\nEnter 'quit' to end the program. "

message = ""
while message != 'quit':
    message = input(prompt)

    if message != 'quit':
    print(message)
```

Using a flag

Flags are most useful in long-running programs where code from other parts of the program might need to end the loop.

```
prompt = "\nTell me something, and I'll "
prompt += "repeat it back to you."
prompt += "\nEnter 'quit' to end the program. "

active = True
while active:
    message = input(prompt)

    if message == 'quit':
        active = False
    else:
        print(message)
```

Using break to exit a loop

```
prompt = "\nWhat cities have you visited?"
prompt += "\nEnter 'quit' when you're done. "

while True:
    city = input(prompt)

    if city == 'quit':
        break
    else:
        print(f"I've been to {city}!")
```

Accepting input with Sublime Text

Sublime Text, and a number of other text editors can't run programs that prompt the user for input. You can use these editors to write programs that prompt for input, but you'll need to run them from a terminal.

Breaking out of loops

You can use the `break` statement and the `continue` statement with any of Python's loops. For example you can use `break` to quit a for loop that's working through a list or a dictionary. You can use `continue` to skip over certain items when looping through a list or dictionary as well.

While loops (cont.)

Using continue in a loop

```
banned_users = ['eve', 'fred', 'gary', 'helen']

prompt = "\nAdd a player to your team."
prompt += "\nEnter 'quit' when you're done. "

players = []
while True:
    player = input(prompt)

    if player == 'quit':
        break
    elif player in banned_users:
        print(f"{player} is banned!")
        continue
    else:
        players.append(player)

print("\nYour team:")
for player in players:
    print(player)
```

Avoiding infinite loops

Every while loop needs a way to stop running so it won't continue to run forever. If there's no way for the condition to become false, the loop will never stop running. You can usually press Ctrl-C to stop an infinite loop.

An infinite loop

```
while True:
    name = input("\nWho are you? ")
    print(f"Nice to meet you, {name}!")
```

Removing all instances of a value from a list

The `remove()` method removes a specific value from a list, but it only removes the first instance of the value you provide. You can use a while loop to remove all instances of a particular value.

Removing all cats from a list of pets

```
pets = ['dog', 'cat', 'dog', 'fish', 'cat',
        'rabbit', 'cat']
print(pets)

while 'cat' in pets:
    pets.remove('cat')

print(pets)
```