Text Input and Conditionals

Text Input

Many programs allow the user to enter information, like a username and password. Python makes taking input from the user seamless with a single line of code:

```
input("How do you get input in Python? ")
```

`input()` will print the string you put in the parentheses and wait for the user to respond. Often, you will use a question as the input string, followed by a space to separate the user's text from the output text. In this example, the program will print "How do you get input in Python? " and user can then enter text to respond.

```
How do you get input in Python? By using input()
>>>
```

The user responded with "By using input()," but right now, that response isn't doing anything. To catch the response and do something with it, we need to set that to a variable.

```
x = input("How do you get input in Python? ")
print(x)
```

Now, the variable **`x`** stores the response from the user.

```
How do you get input in Python? By using input()
By using input()
```

`Try it yourself`: ask for your name and store it in a variable, then print the variable.

Strings and Numbers

Suppose you ask the user for a number, such as a pin number. **`input()`** gives you the user's response. However, it doesn't determine whether or not the user entered a number. Instead, it just gives you a string. So, you can sometimes get errors if you try to treat the input as a number immediately. If you run the following program, Python will give you an error.

```
x = input("Enter a number: ")
print(x + 1)
```

```
Enter a number: 1
Traceback (most recent call last):
  File "C:\Users\Justin Norman\Doc
5, in <module>
    print(x + 1)
TypeError: must be str, not int
```

This is because Python can't add a string and a number together, and **`input("Enter a number: ")`** is a string. Use the **`int()`** function if you want to treat the input as an integer or the **`float()`** function if you want to treat the input as a number with a decimal point. If you are in doubt, just use the **`float()`** function.

```
x = input("Enter a number: ")
x = int(x)
print(x + 1)
```

```
Enter a number: 1
2
```

To convert a number to a string, use **`str()`**.

```
x = 1
x = str(x)
print("This is now a string: " + x)
```

```
This is now a string: 1
```

`Try it yourself:` ask for your age, then print what your age will be in one year.

Truth Statements

Once you have a response from the user, you'll want to tell if that response is correct or not, such as a password that was entered or a button that was pressed. To do so, you'll need to check a property of the response. In the above example, you'll want to check if the user enters the correct answer by making sure "input()" is somewhere in the response. To do this, you'll need to use a **truth statement** or **boolean expression**. A truth statement is a statement that evaluates to a boolean value, either true or false. Here is an example of a truth statement:

```
x = 1
y = 1
print(x == y)
```

`x == y` is a truth statement. The code **==** checks whether or not **`x`** is equal to **`y`**. If it is, it will have the value of **`True`**. If it is not, it will have a value of **`False`**. In the example above, **`x`** is equal to **`y`**, so the output is:

```
True
```

There are several other types of truth statements, like:

- **`x > y`** - is true if x is greater than y
- **`x >= y`** - is true if x is greater than or equal to y
- **`x < y`** - is true if x is less than y
- **`x <= y`** - is true if x is less than or equal to y
- **`x != y`** - is true if x is not equal to y
- **`x in y`** - is true if x is somewhere in y

These work for more than just numbers. Often, they'll work for any data type they make sense for. In fact, they can even be used on other truth statements because they're just boolean values. Just use your intuition: **==** can check if two strings are the same, **<** can check if one string comes before another alphabetically, **>** can check if one string comes after another alphabetically, and so on. When in doubt, experiment using the shell.

```
>>> "Hello" < "Pizza"
True
>>> "Fries" > "French"
True
>>> "I am" == "Am I"
False
>>> "Pizza" in "Pizza Pie"
True
```

You can also combine boolean values, which can be truth statements, using the following keywords:

- **`x or y`** - is true if **`x`** is **`True`**, **`y`** is **`True`**, or both **`x`** and **`y`** is **`True`**
- **`x and y`** - is true if both **`x`** and **`y`** is **`True`**
- **`not x`** - is true if x is not true

For example:

```
>>> True and False
False
>>> True or False
True
>>> not True
False
```

Using a truth statement, we can check if the user's response is correct:

```
x = input("How do you get input in Python? ")
print("input()" in x)

How do you get input in Python? By using input()
True
```

Beware! Keep track of your syntax. A common mistake for programmers is to use **=** instead of **==**. **`x = y`** assigns the value of **`y`** to **`x`**, but **`x == y`** checks if **`x`** and **`y`** have the same value. Using the wrong one can break your program. If you use **`x == y`** when you mean **`x = y`**, the variable will not be assigned properly.

```
x = 1
y == x
print(y)

Traceback (most recent call last):
  File "C:\Users\Justin Norman\Docu
Junk.py", line 17, in <module>
    y == x
NameError: name 'y' is not defined
```

If you use **`x = y`** when you mean **`x == y`**, the code may break or flags will be assigned improperly.

```
x = 1
flag = x = 1
print(flag)

1
```

Be sure you're using the right equals sign!

`Try it yourself:` write a program that asks for your age, and prints **`True`** if you are at least 16 years old.

Conditionals

We can now determine whether the user's answer is correct, we still want to do something that depends on it. For example, you only want the user to log in if the password is correct. You can do this using a **conditional**. A **conditional** is a command that takes a **`True`** or **`False`** value, which can come from a truth statement, and does something depending on that. This is also known as an **if-else statement**.

```
x = input("How do you get input in Python? ")
if "input()" in x:
    print("Correct!")
```

"input()" in x is **True** only if **"input()"** is somewhere in **x**. The line **if "input()" in x** checks the truth statement. If it is **True**, then it executes all the code below it that is indented.

```
How do you get input in Python? By using input()
Correct!
```

You will also want to tell the user if his answer was incorrect. You can do this by placing an **else** directly after the indented code of the **if**. The code indented beneath **else** will run if the condition of the **if** it is placed after is **False**.

```
x = input("How do you get input in Python? ")
if "input()" in x:
    print("Correct!")
else:
    print("Wrong!")

How do you get input in Python? By using print()
Wrong!
```

We now have a working program that asks the user a question and determines whether they got the right answer! But before you get too carried away, make sure you remember that indentation is important. If you don't indent correctly, the whole program may crash or run improperly. For example:

```
x = input("How do you get input in Python? ")
if "input()" in x:
    print("Correct!")
else:
print("Wrong!")
```

This code won't run because Python needs an indented statement after an **if** or an **else**. If the code did run, "Wrong!" would be printed regardless of the boolean value of the truth statement. Consider another example:

```
x = input("How do you get input in Python? ")
if "input()" in x:
    print("Correct!")
else:
        print("Wrong!")
```

This code won't run because the code run by the **`if`** statement must only be indented one more than it. In this case, the **`else`** is indented 0 times and the **`print("Wrong!")`** is indented 2 times, so Python will give an error. Consider one more example:

```
x = input("How do you get input in Python? ")
if "input()" in x:
    print("Correct!")
else:
    print("Wrong!")
print("Try again!")
```

You may want "Try Again!" to print only if the user entered the wrong answer. However, it will always print because it isn't indented.

```
How do you get input in Python? By using input()
Correct!
Try again!
```

So, always be careful with indentation!

`Try it yourself`: write a program that asks for a color and tells you if that color is the same as the your favorite color.

Multiple Conditionals

You may want to check the input against different conditions. With a security program, there may be several correct passwords, all of which are different from each other and take you to different places. You can check this using multiple conditionals. Here is an example:

```
x = input("What is the password? ")
if "Hello" in x:
    print("Hi")
elif "Secret" in x:
    print("Shhhh!")
elif "Flim" in x:
    print("Flam!")
elif "Zim" in x:
    print("Zam!")
else:
    print("That's not the password...")
```

Each password prints a different statement, so we want to check for each one of them. We do so using an **`elif`**. **`elif`** is a hyphenated way of writing "else if" and runs a conditional statement if the previous **`if`** was false. **`elif`** must be placed after an **`if`** or an **`elif`** and can be stacked infinitely. Just like an **`if`**, an **`elif`** does not need to have an else after it.

```
What is the password? Hello
Hi

What is the password? Secret
Shhhh!

What is the password? Flim
Flam!

What is the password? Zim
Zam!

What is the password? Foo
That's not the password...
```

`Try it yourself:` ask for a name and tell the user if that's one of the programmer's first, middle, or last names.

Nested Conditionals

You can put conditionals inside each other to allow for more branching behaviour. Many programs will layer security by requesting a password, then a pin, then a security question. This is known as **nesting**. To nest a conditional, simply place an if statement one more indentation under another if statement, as usual.

```
x = input("Pick a number between 0 and 10: ")
x = int(x)
y = input("Pick a number between 0 and 10: ")
y = int(y)
if x < y:
    if y - x > 5:
        print("The difference is greater than 5")
    else:
        print("The difference is less than 5")
else:
    if x - y > 5:
        print("The difference is greater than 5")
    else:
        print("The difference is less than 5")
```

This program determines whether or not the difference between two numbers is greater than five. To do so, you need to determine which number is larger before performing the subtraction to avoid negative numbers. So, we first check whether or not **`x`** is less than **`y`**. Then, we perform our subtraction based on which number is larger.

```
Pick a number between 0 and 10: 3
Pick a number between 0 and 10: 7
The difference is less than 5

Pick a number between 0 and 10: 10
Pick a number between 0 and 10: 0
The difference is greater than 5
```

Again, be careful with indentation. Code run by a statement must be indented one more than the statement is, regardless of where it is. So, the code inside an **`if`** statement needs to be indented once, and the code inside an **`if`** statement nested in another **`if`** statement must be indented twice.

`Try it yourself:` write a program that asks the user for a password, then a pin number. If the user gets either wrong, tell the user and quit the program.

Flags in Conditionals

It can be annoying to have to write out a long truth statement every time you want to check something. You wouldn't want the user to enter their password each and every time they try to access a website. You can simplify the process by using a **flag**. A flag is a variable that holds a boolean value, either true or false, based on something in the code. With a flag, we can simplify our program one further step:

```
x = input("How do you get input in Python? ")
correct = "input()" in x
if correct:
    print("Correct!")
else:
    print("Wrong!")
```

This will perform the same task as before, except now, we don't need to type **`"input()" in x`** every time we want to check if the answer is correct. Instead, we can simply check the value of **`correct`**.

```
How do you get input in Python? By using input()
Correct!
```

`Try it yourself:` change any of the programs you've written to use flags.

Exercises

1. Create a program that asks the user for their birthday and tells them their zodiac sign.

2. Create a program that makes a user go through some security checks before telling him your name.
3. Create a small game of twenty questions (it doesn't actually need to be twenty questions).
4. Create a small text adventure, where the program presents the user with situations, asks the user what he or she wants to do, and changes the story accordingly. You may either provide them with a list of choices to choose from, or you may allow them to type their choices.