Notes - Sunday 12/20/20

Starting! My 2 week goal is to create the game Snake in Python. So I can kickstart my new game dev studio.

I got overwhelmed with options for editors and went with Visual Studio Code ("VS Code") which was recommended by the jwaz nerd chat. Mu will be my backup as I'm currently not sure how VSCode will work with the Pybadge or how to get a REPL working.

Below is an example code we were to use to teach us how to save a .py file in Sublime or another text editor.

print('hello world')

Jupyter Notebooks Notes

How to open files

I can open Jupyter notebooks through the actual Jupyter Lab program by running `jupyter-lab` in the CLI. That will open something up in the browser which I can navigate around in. Or I can open the notebooks up in VSCode which I like since the one app can open up my code, my own notebooks, and the notebooks from the class.

How to create new notebooks in VSCode

`Ctrl+Shift+P` then type in "Create new blank Jupyter notebook

Git Notes

Once I start getting more complicated projects, I'll want to keep backups or setup "source code management" (SCM) to track changes to my code. Git helps to do this and is seperate from Github. Git can be a local tool and I can back up my projects and code to my Nextcloud server if I want. Github is a tool that uses git but is onilne and I could share projects with others among other features.

As of 12/20/20. I haven't setup git yet. I should probably do so in the future.

Some helpful links:

- VSCode documentation
- Colt Steele's Learning about Git in 15 mins
- Colt Steele notes to go with Youtube video above
- Digital Ocean CheatSheet

Types of Data

- Integers (int) = Whole numbers - 3, 300, 200
- Floating Point (float) = Numbers with decimal places - 2.3, 4.6, 100.0
- Strings (str) = Ordered sequence of characters so just text - "hello", "Sammy", "2000"
- Lists (list) = Ordered sequence of objects so text, Integers, or Floating Points could go in here - [10,"hello",200.3]
- Dictionaries (dict) = Unordered Key:value pairs (more info coming later; currently doesn't make sense to me) - {"mykey":"value", "name":"Anson"}
- Tuples (tup) = It's said like "too-pul-es". It's a sequence of objects that and the order can't be changed aka immutable (more info coming). - (10,"hello",200.3)
- Sets (set) = Unordered collection of unique objects {"a","b"}
- Booleans (bool) = Logical value indicating True or False

Note that:

Integers do not have decimals.

A Floating Point number can display digits past the decimal place. A floating point number does not represent EXACT values, just a close/increasingly better approximation of a value. For example 1/3 doesn't mean the value is 0.3, 0.33, or 0.333. It's how the computer holds the approximate value.

So 0.1+0.2-0.3 doesn't equate to 0.0. Just a value really close to 0. Click this link for more info.

Numbers in Python

There's 2 main types of numbers as listed above: integers and floating point.

Python can be used as a straight up calculator. I can literally type in 2+2 and run the code to get a value.

Mod operator %

You could also have python output a "Mod operator" or a "Modulo".

Let's say you wanted to do 7/4. You would say that 4 goes into 7 one time with a remainder of 3. So the mod operator will output the remainder. In this way, you can quickly see if a number is easily divisible by another number.

In the previous example, we had a remainder of 3. So 4 does not go into 7 easily. But 5 goes into 50 cleanly. We would expect the remainder to be 0.

To use the mod operator, you'd use the % sign. `X % Y`

You can also use the mod operator to check if a number is even or odd. For example if you have a variable that is a number and you're not sure if it's even or odd. If I get a remainder from the mod, then it's odd. If I get 0, then it's even.

23 is odd. 20 is even.

Exponents **

I can do Exponents by doing `x ** y`.

Order of Operations

Remember PEMDAS for order of operations - parenthesis, exponents, multiplication/division, addition/subtraction

If I just type in normal formula. Python will follow the order of operations. If I want to control the order Python does the math in, I will have to use parenthesis.

Variable Assignments

We could assign the above data types a variable name.

Names can't:

- Start with a number
- No spaces - need to use underscore _ instead
- No special symbols besides _ (Python will complain if the special symbols are used)

Names should:

- Be in lowercase as a best practice
- Avoid keywords as variable names such as list or str (the editor should highlight those keywords a different color)

Python uses dynamic typing. So variable names can be reassigned. The below can happen. In other langauges such as C++, this can't be done. Once the integer of "my_dogs" is assigned, it stays that way.

- `my_dogs = 1`
- ``my_dogs = ["Armani"]

Dynamic Typing Pros:

- easy to work with
- saves time since you don't have to assign a data type. So for example, in the above example, C++ wants to see an integer data type assignment. `int my_dogs=1`

Dynamic Typing Cons:

- Easy to have unexpected data types could cause bugs

How to determine data type using Type()

Let's say we don't know what kind of data "a" is and we want to find out. We would use `type(a)`

- type() could help find those errors (discussed below)

I ran the code below in a Jupyter notebook.

```
a = 5

type(a)

a = 30.1

type(a)
```

Syntax Highlighting and variables

Note that above we can see that type() shows that my data was an integer then a floating point number. We can also see that "type" is highlighted in an aqua green color. That shows that its a

python keyword. So we shouldn't name a variable "type". If I accidently make a variable named "type", I can undo this by restarting the variable. In VSCode, the symbol is the little green circular arrow.

Variable name example

Let's say I want to calculate my taxes.

- Income = 100
- Tax Rate = 0.1

```
my_income = 100


tax_rate = 0.1


my_taxes = my_income * tax_rate
```

Then run `my_taxes` and the answer should be 10.0

Strings

Strings are sequences of characters. The syntax used is single or double quotes.

- 'hello'
- "hello"
- " I like pie "

Strings are ordered sequences. So we can index and slice to grab parts of the string. The spaces/whitespace also count as characters. The syntax is ' or " so that if there's an abbreviation for example, you can include the abbreviation without confusing python. So for example the string would be:

- "I'm a lover of pie"

Instead of:

- 'I'm a lover of pie' Which would generate an "invalid syntax" error if the code was run.

Indexing

Indexing allows you to grab a single character from the string. It uses [] notation after the string or string variable. If my string is hello:

- 'hello'
- Characters = h e l l o
- Index show = 1 2 3 4 5

Reverse index allows me to grab the last letter of hello even if I don't know how long the string is.

- Reverse Index = 0 -4 -3 -2 -1

Slicing

Slicing allows me to slice a part of the string to grab a subsection of multiple characters. The syntax would be:

- [start:stop:step]

Where:

- start = numerical index for the start of the slice
- stop = index I'll go up to but not include
- step = size of the slice we'll take (size of the jump)

```
'hello world'
```

So in the above examples, we're asking Python to return the string we type in. That's why we see the little quotes. But we could also print the strings out. This removes the quotes and actually prints out the string. If we wanted to print multiple strings for example, we would need to use the print command.

In the print command, we could also use escapes `\n` or tabs `\t` to change the formatting of the thing we print out. Syntax would be `print("hello \n world"` and `print(hello \t world)` respectively. More on this coming up.

We can also check the length of the string using the `len` command. Syntax is `len('x')`

String Indexing and Slicing - Let's look at it more in depth Indexing in more depth Let's set a variable called mystring to be a sting that says "Hello World". The syntax for using a string would be mystring[position]. The positions start at 0, NOT 1. When we type in the variable, this is referred to as "calling". So below I'll set the mystring variable then call it and run an index to get the first letter H which is in position 0. Typing in the 0 would be called "passing".

If the position 0 gives us H. The index would count left to right in positive integers. So let's say I want the capital W. I would pass 7 in the index to get it.

Let's say I want to have "d" be the result of calling the index. I could use the normal index and pass 10. Or I could use the reverse index. If I want to use a reverse index, I'd count from right to left from the end of the string. So I would pass -1 to get the "d" or -5 for the "W".

Slicing in more depth

Let's say now `mystring = 'abcdefghijk"`. I need to redefine mystring. Then use syntax `mystring[position:]` . The : shows where the silce should start or end. I want to grab a slice or chunk of the string that's C through to the end.

So to grab C through to the end I'd type `mystring[2:]`.

If I want to grab "abc", I'd write `mystring[:3]`.

I can also grab a slice in the middle so if I want "def" I'd type `mystring[3:6]`

Remember that you're going UP to the position but NOT including that position. So that's why I'm including position 3 which id "d" in my string but it won't print out "abcd" but instead prints "abc".

Revision #5
Created 21 December 2020 00:31:39 by cba88
Updated 22 December 2020 18:14:19 by cba88