DAY-2

Syntax, Variables, and Numbers

Run this line on your editor-

```
print("You have come this far, so you are special.")
```

Output - You have come this far, so you are special.

Print is a Python function that displays the value passed to it on the screen. We call functions by putting parentheses after their name, and putting the inputs (or *arguments*) to the function in those parentheses.

2.1 variable.py

```
# Try_ something  challenging & learn a lot at once.
Flag = 0
Flag = Flag + 4

if Flag > 0:
    print("It is a positive number")

Multi = "Multi " * Flag
print(Multi)
```

There's a lot to unpack here! This silly program demonstrates many important aspects of what Python code looks like and how it works. Let's review the code from top to bottom.

Variable assignment: Here we create a variable called Flag and assign it the value of 0 using =, which is called the assignment operator.

Aside: If you've programmed in certain other languages (like Java or C++), you might be noticing some things Python *doesn't* require us to do here:

- we don't need to "declare" Flag before assigning to it
- we don't need to tell Python what type of value “Flag” is going to refer to. In fact, we can even go on to reassign “Flag” to refer to a different sort of thing like a string or a Boolean.

The first line above is a **comment**. In Python, comments begin with the # symbol.

Next we see an example of reassignment. Reassigning the value of an existing variable looks just the same as creating a variable - it still uses the = assignment operator.

In this case, the value we're assigning to “Flag” involves some simple arithmetic on its previous value. When it encounters this line, Python evaluates the expression on the right-hand-side of the = (0 + 4 = 4), and then assigns that value to the variable on the left-hand-side.

**Python is prized for its readability and the simplicity. ☺☺

Note how we indicated which code belongs to the if. "It is a positive number." is only supposed to be printed if “Flag” is positive. But the later code (like print(Multi)) should be executed no matter what. How do we (and Python) know that?

The colon (:) at the end of the if line indicates that a new "code block" is starting. Subsequent lines which are **indented** are part of that code block. Some other languages use {curly braces} to mark the beginning and end of code blocks. Python's use of meaningful whitespace can be surprising to programmers who are accustomed to other languages, but in practice it can lead to more consistent and readable code than languages that do not enforce indentation of code blocks.

The later lines dealing with “Multi” are not indented with an extra 4 spaces, so they're not a part of the if's code block. We'll see more examples of indented code blocks later when we define functions and using loops.

```
"It is a positive number"
```

Strings can be marked either by double or single quotation marks. (But because this particular string *contains* a single-quote character, we might confuse Python by trying to surround it with single-quotes, unless we're careful.)

```
Multi = "Multi " * Flag
print(Multi)
```

Output: Multi Multi Multi Multi

The * operator can be used to multiply two numbers (3 * 3 evaluates to 9), but amusingly enough, we can also multiply a string by a number, to get a version that's been repeated that many times. Python offers a number of cheeky little time-saving tricks like this where operators like * and + have a different meaning depending on what kind of thing they're applied to.

Numbers and types in Python

```
Flag = 0
```

"Number" is a fine informal name for the kind of thing, but if we wanted to be more technical, we could ask Python how it would describe the type of thing that Flag is:

```
type(Flag)
```

Output – int

```
type(3.8)
```

Output- float

A float is a number with a decimal place - very useful for representing things like weights or proportions.

type() is the second built-in function we've seen (after print()), and it's another good one to remember. It's very useful to be able to ask Python "what kind of thing is this?"

A natural thing to want to do with numbers is perform arithmetic. We've seen the + operator for addition, and the * operator for multiplication (of a sort). Python also has us covered for the rest of the basic buttons on your calculato

Operator	Name	Description
a + b	Addition	Sum of a and b
a - b	Subtraction	Difference of a and b
a * b	Multiplication	Product of a and b
a / b	True division	Quotient of a and b
a // b	Floor division	Quotient of a and b , removing fractional parts
a % b	Modulus	Integer remainder after division of a by b
a ** b	Exponentiation	a raised to the power of b
-a	Negation	The negative of a

```
print(5 / 2)
print(6 / 2)
```

Output –
2.5
3.0

It always gives us a float.

The // operator gives us a result that's rounded down to the next integer.

```
print(5 // 2)
print(6 // 2)
```

Output: 2
Output: 3

Order of operations

The arithmetic we learned in primary school has conventions about the order in which operations are evaluated. Some remember these by a mnemonic such as **PEMDAS** - **P**arentheses, **E**xponents, **M**ultiplication/**D**ivision, **A**ddition/**S**ubtraction.

Python follows similar rules about which calculations to perform first. They're mostly pretty intuitive.

```
8 - 3 + 2
```

Output: 7

```
-3 + 4 * 2
```
Output: 5

2.2. Do yourself

```
hat_height_cm = 25
my_height_cm = 190
# How tall am I, in meters, when wearing my hat?
total_height_meters = hat_height_cm + my_height_cm / 100
print("Height in meters =", total_height_meters, "?")
```
Output: Height in meters = 26.9 ?

Parentheses are your useful here. You can add them to force Python to evaluate sub-expressions in whatever order you want.

```
total_height_meters = (hat_height_cm + my_height_cm) / 100
print("Height in meters =", total_height_meters)
```

Output : Height in meters = 2.15

Builtin functions for working with numbers

min and max return the minimum and maximum of their arguments, respectively...

```
print(min(1, 2, 3))
```
Output: 1

```
print(max(1, 2, 3))
```
Output: 3

abs returns the absolute value of it argument:

```
print(abs(32))
```
Output: 32

```
print(abs(-32))
```
Output: 32

In addition to being the names of Python's two main numerical types, int and float can also be called as functions which convert their arguments to the corresponding type:

```
print(float(10))
```
Output : 10.0

```
print(int(3.33))
```
Output: 3

 # They can even be called on strings!

```
print(int('807') + 1)
```
Output: 808

Greater Than
55 > 22

Less Than
132 < 123

2.3. Area of Circle

```
pi = 3.14159 # approximate
diameter = 3

radius = diameter / 2
area = pi * radius**2
print(area)
```

Output: 7.0685

2.4. Swap

```
a = [1, 2, 3]
b = [3, 2, 1]

temp = a
a = b
b = temp
print(a, b)
```

Output: [3,2,1] [1,2,3]

You can also use swap(a, b)

Have a look on exercise code, run this -
2.5 Exercise Code:

```
mystring = "hello"
myfloat = 10.0
myint = 20

if mystring == "hello":
    print("String: %s" %mystring)
if isinstance(myfloat, float) and myfloat == 10.0:
    print("Float: %f" %myfloat)
if isinstance(myint, int) and myint == 20:
    print("Integer: %d" %myint)
```

Output: String: hello
Float: 10.000000
Integer: 20

----------THANK YOU----------