python basics

week 1

Komal Kaur

agenda

- variables and functions
- conditional operators
- loops
- lists
- disclaimer: workshop intended for beginners!

variables and functions

- variables store information
- data types: int, float, str, bool
- functions are what do/change things in a program
  - input(s) and output(s)
  - defined and called

user-defined functions

- user-defined: a function you wrote yourself

```
def function():
    # code code code
    return value
```

`def` keyword tells the compiler that the following is a function

notice the indentation: the body of the function (including the return statement) is indented under the definition of the function

`return` keyword tells the compiler that the following is what the function evaluates to

math functions

- python has the following syntax for math operations:

Operation	Notation	Example
add	+	1 + 1 “hello” + ”world”
subtract	-	50 - 8
Multiply	*	4 * 3
Divide	/	60 / 5
Exponentiate	**	5 ** 2
Modulo	%	4 % 3
Floor divide	//	10 // 8

conditional operators

- conditional statements are based on boolean values (`True` or `False`)
- operators are `if`, `elif` (else if), and `else`
- structure:

```
if first_condition:
    do_this()
elif second_condition:
    do_this_thing()
else:
    do_this_instead()
```

the conditions are booleans. if they are true, then the program enters the indexed part and runs the code inside. if the condition is false, the indented code is not run and the program moves on to the next section.

`if` statements do not need `elif` or `else` statements but the `elif` and `else` statements MUST have an `if` statement before them.

It takes some practice to know which way to order the conditions, but in general it's better to put the "tightest" restriction first and the least specific case as the else case.

`else` statements do not have a conditional boolean to evaluate. The `else` always runs unless the `if` or `elif` ran.

loops

- code inside the loop runs for a set number of times
- once the loop is done running, the code picks up where it left off and continues as normal
- good for repetitive tasks
- different types of loops:
  - while loop
  - for loop

loops

while loop

- this runs **while the condition is true**
- once the condition is false the program exits the loop

```
while condition:
    do_this()
    now_do_this()
```

note python's indentation. the body of the loop is indented under it. anything not in the loop has the same indentation level as the definition of the loop.

while the loop is running, the `do_this` function is called. once the condition is false, the loop is exited and `now_do_this` is called.

loops

for loop

- the for loop sets the number of times the loop runs at the beginning and does not depend on a condition

```
for i in range(5):
    do_this()
now_do_this()
```

```
for i in list:
    do_this()
now_do_this()
```

note: whatever follows the `in` has to be something we can iterate through

lists

AKA arrays in other languages

- can store multiple data points of different types
  - ex: a list can consist of a variable, a float, and a bool
  - lists can store lists
  - lists can store themselves (very cool and weird)
- lists have their own methods to be able to interact with them in a useful way
  - append, pop, remove, insert, count, reverse, len
- syntax to define a list:

```
name = [element1, element2, element3]
```

lists

traversals

- traversing a list = going through a list
- indexing in programming: the first element the 0th one, not the 1st one.
  - ex: the item at index 1 in the list [a, b, c, d] is b
- splicing: getting a view of a portion of a list

```
list[start_index: end_index: step]
```

- start is inclusive, end is exclusive, step is how many at a time to return