WORKSHEET 3
Conditionals

3.1 Booleans

- Assume any test is either `True` / `False`
- There is **default value that returns to False** (0 value, empty string), and others return True
  - `print(bool(1))`
  - `print(bool(-2.0))`
  - `print(bool("False"))`

```
True
True
True
```

  - `print(bool(0))`
  - `print(bool(0.0))`
  - `print(bool(""))`
  - `print(bool(False))`

```
False
False
False
False
```

3.2 Relational Operators

- For int and float

Python	**Meaning**	**Math Notation**
<	less than	<
<=	less than or equal	≤
==	equal	=
!=	not equal	≠
>	greater than	>
>=	greater than or equal	≥

- `print(1 > 2)`
- `print(1 + 1 >= 1)`
- `print(2.0 == (4 / 2.0))`

```
False
True
True
```

3.3 String Comparisons

- For string
- Letters are sorted by **alphabetical order** ***lower case > upper case**
  - `print('he' < 'hi')`
  - `print('Hell' >= 'Hello')`
  - `print('h' > 'H')`
  - `print('Z' < 'a')`

```
True
False
True
True
```

- characters have numbers associated with them
  - `print(ord(‘A’))`

3.4 Substrings

- Whether a string is in another string (case sensitive)

-
  ```
  print('Hell' in 'Hello')
  True
  ```

3.5 Logical Operators

- Binary logical operators (apply 2 Boolean variables)
  - `and`
  - `or`
- Unary logical operator (for 1 Boolean variable)
  - `not`

Operands		Logical Operator	
A	B	A and B	A or B
False	False	False	False
True	False	False	True
False	True	False	True
True	True	True	True

-
  - `print(True and True)`
  - `print(True and 1 != 1)`
  - `print(1 > 2 or True)`
  - `print(not True)`

```
True
False
True
False
```

3.6 Order of operators

- Relational operators (including in) -> not -> and -> or
  - `print(not 1 > 2 and 1 > 0 or "din" in "coding")`
    - `not False and True or True`
    - `True and True or True`
    - `True or True`
    - `True`

3.7 Conditional Blocks

-
  ```
  if <condition>:
        <block of statements>
  ```
  - e.g.
    - `n = int(input("Enter an integer: "))`
    - `if 0 < n < 6:`
    - `    print('You entered a positive integer less than six.')`
    - `print('Try again with another integer!')`

- To decide between alternatives:

-
  ```
  if <condition>:
        <first block of statements>
  ```

-
  ```
  elif:
        <second block of statements>
  ```

-
  ```
  else:
        <alternative block of statements>
  ```

- ^ at the start of the string (**prefix**) and $ at the end of the string (**suffix**)
    - `sci='^' + sci`
    - `sci= sci + '$'`

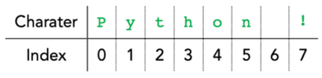
        -
          ```
          elif ('^comp' in sci) or ('^info' in sci):
                print("Computing ftw!")
          ```

        -
          ```
          elif ('y$' in sci):
                 print("Au naturel!")
          ```

WORKSHEET 4
Sequences

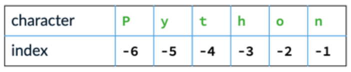
4.1 Strings as sequences

- Python numbers the position of each character within a string, starting with the first character at position number 0
  - Includes spaces and full spots



Charater	P	y	t	h	o	n		!
Index	0	1	2	3	4	5	6	7

  - Can work from end of string (-1 index last character)



character	P	y	t	h	o	n
index	-6	-5	-4	-3	-2	-1

4.2 Indexing

- Access a particular character at a particular position

```
s = "The number is 42."
print(s[0])
print(s[1])
T
h
```

- Find length of strings

```
s = "The number is 42."
n = len(s)
print(n)
print(len("Hello"))
17
5
```

4.3 Slicing (Subscripting)

- Access certain part of substring
  - if the start index is 0 then you can leave it blank
  - if the end index is the length of the string then you can leave it blank
  - **does not include last index**

```
s = "The number is 42."
print(s[:5])
print(s[5:])
print(s[:])
The n
umber is 42.
The number is 42.
```

4.4 Slicing with steps and direction

- Third no. when slicing indicate how many steps to through the list
- **If -1, direction of slice changes**
  - If beyond string -> empty string returned

```
s = "abcdef"
print(s[::2])
print(s[2::-1])
print(s[2:0:-1])
ace
cba
cb
```

4.5 Lists

- Splicing / concatenated techniques can be applied
- Empty list:
    - `Empty = [ ]`

    - `my_words = ['pig', 'pineapple', 'panoply', 'polyp']`
    - `my_costs = [5.0, 12.0, 200000000.59]`
    - `my_jumble = ['jumbly', 4, 'wumbly', 'number', 5]`
    - `print(my_costs)`
    - `[5.0, 12.0, 200000000.59]`

4.6 Tuples

- Same as list ^, but **immuatable** (cannot be changed after creation)
- Empty tuples
    - `Empty = ( )`
- Nested tuples (second index is to get the element we want from that nested sequence)
    - `my_tuple= ('name', 3, ['a', 'nested', 'list'], 'age')`
    - `print(my_tuple[2])`
    - `print(my_tuple[2][1])`

    ```
    ['a', 'nested', 'list']
    nested
    ```