Intro to Programming Week 2

2-26-2019

Sign In!

http://tinyurl.com/CADIntroPySp19-2

Last Week’s Review

```
print(1 - 4)


last = “Yu”
first = “Sean”
full_name = first + last
print(full_name)
```

```
x = 13
y = 19
x = y - x
y = y - x
print(x)
print(y)
```

Last Week's Review

```
print(1 - 4)
```

OUTPUT:

```
-3
```

```
last = "Yu"
first = "Sean"
full_name = first + last
print(full_name)
```

OUTPUT:

```
SeanYu
```

```
x = 13
y = 19
x = y - x
y = y - x
print(x)
print(y)
```

OUTPUT:

```
6
13
```

Number Operators

- Addition: `+`
- Subtraction: `-`
- Multiplication: `*`
- Division: `/`
- Exponent: `**`

Number operators are performed with the PEMDAS rules.

```
Input: print(4 + 2 * (6 + 1))
Output: 18
```

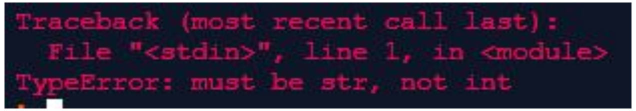
Using the `+` operator on strings yields a different behavior - **concatenation**.

Strings

- Strings are a series of characters. The content of the string must be written between either double or single quotation marks.
  - `“”` or `‘’`
  - `stringExample = “I am a string!” OR ‘I am a string!’`
- String Concatenation: a way for you to combine strings together!
  - `“hello” + “world” = “helloworld”`
  - `x = “hookem”    y = “horns”    z = x + y`
    - `z` now stores `“hookemhorns”`

Strings (cont’d)

- You can **not** add strings to other data types
  - `40 + “acres”` would result in an error
  - `x =“40” + “acres”`
  - `print(x) -> “40acres”`
- You can change the type of a variable using the following built in functions:
  - `int(), str(), bool(), float()`
  - `x = 40`
  - `y = ‘acres’`
  - `print(str(x) + y) -> ‘40acres’`



```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
```

String Concatenation Exercises

1.
   ```
   x = 5 + 5
   y = “eyes_of” + “texas”
   z = str(x) + y
   ```

2.
   ```
   con = 5 + “five” + 8
   cat = 5
   concat = con + cat
   ```

3.
   ```
   cad = str(5 + 8) +
   “three”
   ```

String Concatenation Exercises Answers

1.
   ```
   x = 5 + 5
   y = "eyes_of" + "texas"
   z = str(x) + y
   ```

2.
   ```
   con = 5 + "five" + 8
   cat = 5
   concat = con + cat
   ```

3.
   ```
   cad = str(5 + 8) + "three"
   ```

1.
   ```
   x = 10
   y = "eyes_oftexas"
   z = "10eyes_oftexas"
   ```

2.
   ```
   error
   ```

3.
   ```
   cad = "13three"
   ```

Boolean Operators

- `==` (equals) – True if both are same
- `and` – True only if all are True
- `or` – False only if all are False (True if at least one is True)
- `not` – flips the truth value

Just like PEMDAS rules, you want to always evaluate in the order:

```
not, ==, and, or
```

Boolean Operators Exercises

```
1.  True or True = ?
2.  True and False = ?
3.  True == False = ?
4.  not False = ?
```

```
1.  True or False and True
2.  not False and True
3.  True == False and not False
```

Boolean Operators Exercises Answers

1. `True or True =` **`True`**
2. `True and False =` **`False`**
3. `True == False =` **`False`**
4. `not False =` **`True`**

1. `True or False and True =` **`True`**
2. `not False and True =` **`True`**
3. `True == False and not False =` **`False`**

Lists

- Lists are a collection of items in a particular order.
- The elements are denoted between square brackets: `[]`
  - EX: `name_list = [“Sean”, “Claire”]`
  - You can also declare an empty list: `empty_list = []`
- You can store all kinds of data types (integers, strings, objects, etc.)

Accessing List Elements

- Lists are indexed starting at 0
- EX: `list = [4,2,3,1,0]`

position	0	1	2	3	4
element	4	2	3	1	0

- You can access an element by specifying its position.
  - EX: `list[0]` represents 4

- You can also reference the back of the list using negative numbers
  - `list[-1]` represents 0

```
1.  list[3] = ?
2.  list[-1] = ?
3.  list[-4] = ?
4.  list[6] = ?
```