Variables in Python

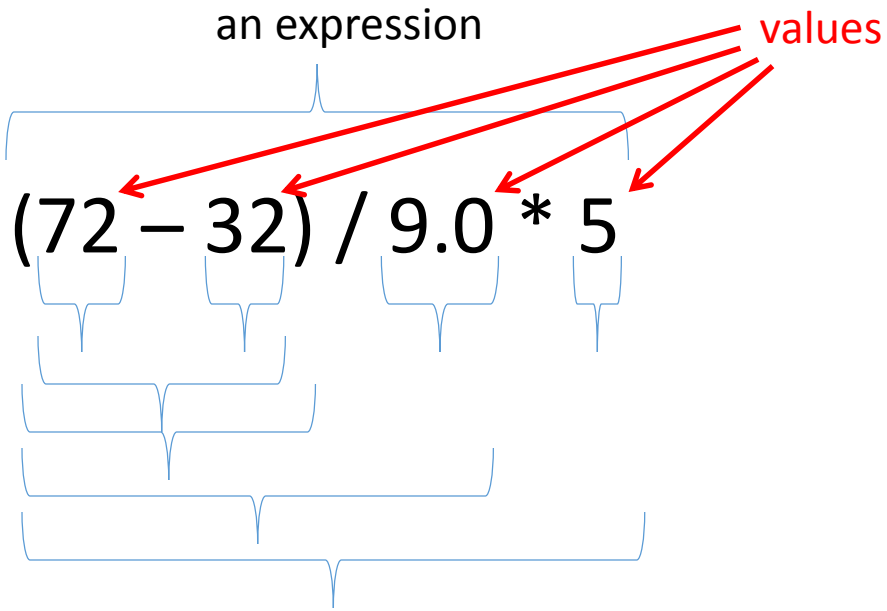
Presented by:-
Honey Pasricha
Sr. Technical Manager
Eklavya Academy Aim to Achieve
www.ekaim.in

You type *expressions*.
Python computes their *values*.

- 5
- 3+4
- 44/2
- 2**3
- 3*4+5*6
  - If precedence is unclear, use parentheses
- (72 – 32) / 9 * 5

An expression is evaluated from the inside out

- How many expressions are in this Python code?



(**72** – **32**) / **9.0** * **5**

(**40**) / **9.0** * **5**

40 / **9.0** * **5**

4.44 * **5**

22.2

Another evaluation example

(72 – 32) / (9.0 * 5)

(40) / (9.0 * 5)

40 / (9.0 * 5)

40 / (1.8)

40 / 1.8

22.2

2. A variable is a container

Variables hold values

- Recall variables from algebra:
  - Let x = 2 …
  - Let y = x …
- To assign a variable, use "*varname* = *expression*"

```
pi = 3.14
pi
avogadro = 6*10**23
avogadro
22 = x                    # Error!
```

No output from an assignment statement

- Not all variable names are permitted

Variable Naming Rules

Although you are allowed to make up your own names for variables, you must follow these rules:

- You cannot use one of Python's key words as a variable name. (See Table 1-2 for a list of the key words.)
- A variable name cannot contain spaces.
- The first character must be one of the letters a through z, A through Z, or an underscore character (_).
- After the first character you may use the letters a through z or A through Z, the digits 0 through 9, or underscores.
- Uppercase and lowercase characters are distinct. This means the variable name `ItemsOrdered` is not the same as `itemsordered`.

Changing existing variables (“re-binding” or “re-assigning”)

```
x = 2
x
y = 2
y
x = 5
x
y
```

Changing existing variables ("re-binding" or "re-assigning")

```
x = 2
x
y = x
y
x = 5
x
y
```

- "=" in an assignment is *not* a statement or promise of eternal equality
- Evaluating an expression gives a new (copy of a) number, rather than changing an existing one

How an assignment is executed

More expressions: Conditionals

```
22 > 4
22 < 4
22 == 4
x = 100          # Assignment, not conditional!
22 = 4           # Error!
x >= 5
x >= 100
x >= 200
not True
not (x >= 200)
3<4 and 5<6
4<3 or 5<6
temp = 72
water_is_liquid = temp > 32 and temp < 212
```

Numeric operators: `+`, `*`, `**`

Boolean operators: `not`, `and`, `or`

Mixed operators: `<`, `>=`, `==`

More expressions: strings

A string represents text

```
'Python'
myclass = "CSE 190p"
""
```

Empty string is not the same as an unbound variable

Operations:

- Length:

```
len(myclass)
```

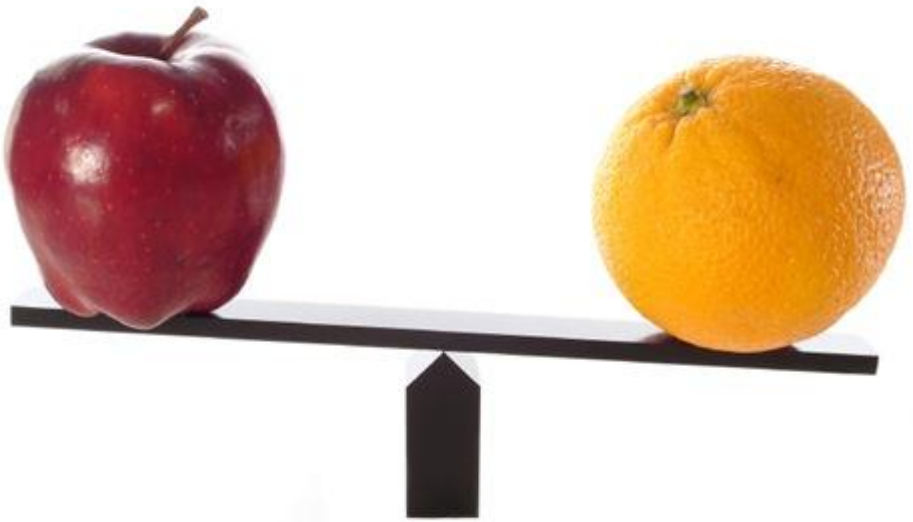
- Concatenation:

```
"Michael" + 'Ernst'
```

- Containment/searching:

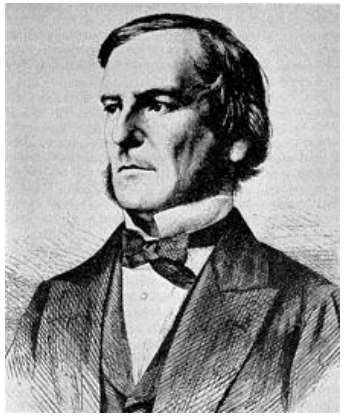
```
'0' in myclass
"O" in myclass
```

3. Different types cannot be compared



Types of values

- Integers (**`int`**): **`-22`**, **`0`**, **`44`**
  - Arithmetic is exact
  - Some funny representations: **`12345678901L`**
- Real numbers (**`float`**, for “floating point”): **`2.718`**, **`3.1415`**
  - Arithmetic is approximate, e.g., **`6.022*10**23`**
  - Some funny representations: **`6.022e+23`**
- Strings (**`str`**): **`"I love Python"`**, **`""`**
- Truth values (**`bool`**, for “Boolean”): **`True`**, **`False`**



George Boole

Operations behave differently on different types

```
3.0 + 4.0
3 + 4
3 + 4.0
"3" + "4"
3 + "4"                          # Error
3 + True                    # Insanity!
```

Moral: Python *sometimes* tells you when you do something that does not make sense.

Operations behave differently on different types

```
15.0 / 4.0
15 / 4                    # Insanity!
15.0 / 4
15 / 4.0
```

Type conversion:

```
float(15)
int(15.0)
int(15.5)
int("15")
str(15.5)
float(15) / 4
```