CIS 1100

Types & Variables! (Lecture)

Python

Fall 2024

University of Pennsylvania

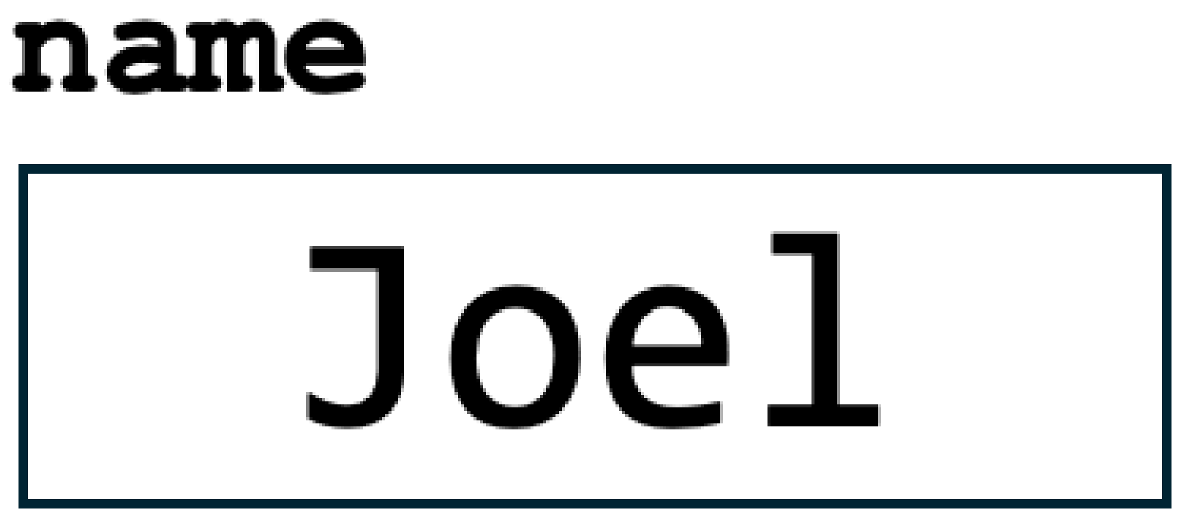
Review: Variables

Let’s start with a simpler example:

```
name = "Joel"
```

What this does is it creates a ***Variable*** named "name" holding the value `"Joel"`.

You can think of a variable as being similar to a box with a name attached to it.



name

Joel

Review: Variables

Let’s start with a simpler example:

```
name = "Joel"
```

The value within the box can change over the lifetime of the program (i.e. while it is being run or executed).

However, it can only hold one thing at a time.

Printing Variables

We can print variables, similar to how we print strings:

```
name = "Joel is aight."
print(name)  # prints "Joel is aight."
```

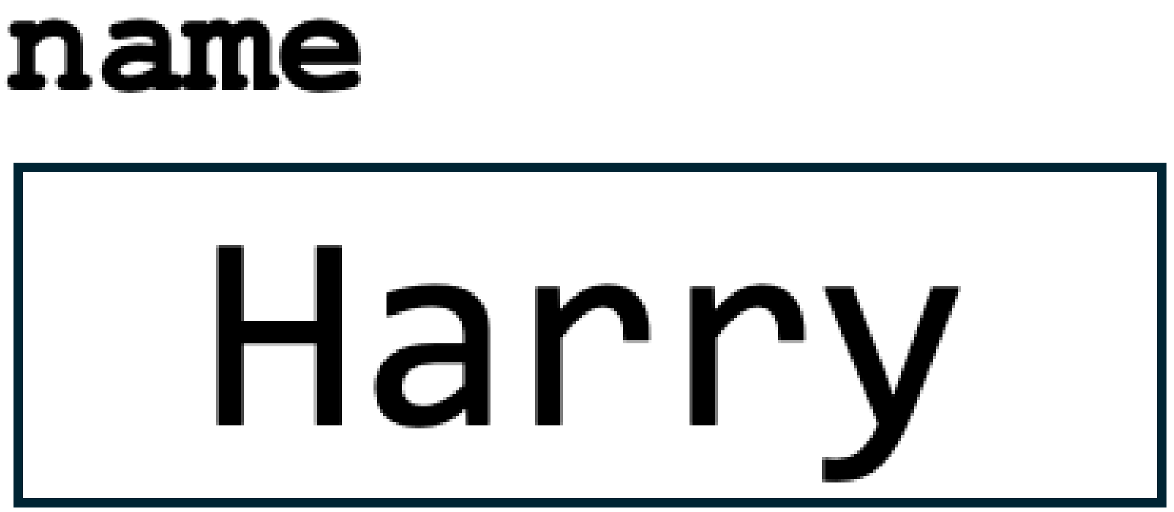
When we pass a variable into a function this tells python to see what is within "that box".

Variables Change

Variables can change over the course of a program, and can only hold one value.

Consider:

```
name = "Harry" # line a  <-----
name = "Joel"  # line b
print(name)
```



name

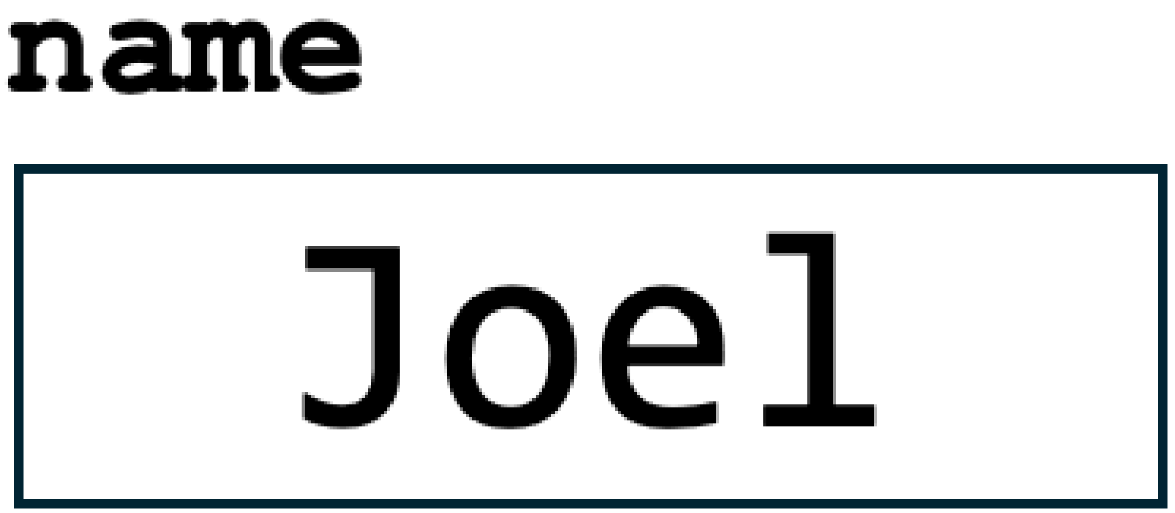
Harry

Review: Variables Change over Time

Variables can change over the lifetime of a program, and can only hold one value.

Consider:

```
name = "Harry" # line a
name = "Joel"  # line b     <-----
print(name)
```



Lecture Activity

S7:

How many values is `fruit` set to by the time we call the `print()` function?

Consider:

```
fruit = "apple"
fruit = "banana"
veggie = "ew"
veggie = "lettuce"
fruit = "pear"
fruit = "tomato" # yes I know, bite me.
print(fruit)     # printing here!
fruit = "Pitaya"
```

Lecture Activity

S7:

How many values is `fruit` set to by the time we call the `print()` function?

Consider:

```
fruit = "apple"
fruit = "banana"
veggie = "ew"
veggie = "lettuce"
fruit = "pear"
fruit = "tomato" # yes I know, bite me.
print(fruit)     # printing here!
fruit = "Pitaya"
```

We want to enforce early on that code runs sequentially.

From top to bottom. One line at a time.

Lecture Activity: Variable Naming and Style

For best practices We follow `lower_snake_case` when naming variables.

M1:

Consider the following variables. Which are both legal (correct python syntax) and follow `lower_snake_case`?

a) `local_counter3`

b) `exec_and_fork`

c) `AwesomeVariable`

d) `awesome_Variable`

e) `import`

Why can't we use `import`?

As we gain more experience in python, we'll see that there are a couple of special words that are reserved for special purposes.

These are called *keywords*.

As mentioned before, we use an `import` keyword to tell the computer we would like to use `Penn Draw` in our python program.

```
import penndraw as pd # 'import' and 'as' are two key words here
pd.line(1, 1, 0, 0)
pd.point(.5, .75)
pd.run()
```

Why can't we use `import`?

Although, it's not important to know all the keywords *now*, here they are.

Note: we've already seen `as` and `import` as keywords!

`False`	`await`	`else`	`import`	`pass`
`None`	`break`	`except`	`in`	`raise`
`True`	`class`	`finally`	`is`	`return`
`and`	`continue`	`for`	`lambda`	`try`
`as`	`def`	`from`	`nonlocal`	`while`
`assert`	`del`	`global`	`not`	`with`
`async`	`elif`	`if`	`or`	`yield`

Review: Expressions

- **Expressions** are portions of a program that have or evaluate to a value.
- Basic expressions are composed of **literals**, **variables**, and **operators**

Term	Definition	Example
Literal	A part of an expression that has a value which can be interpreted *literally*	`4.0` or `"python"`
Variable	A named portion of memory that stores some value	`year`, `x`, or `last_name`
Operator	A symbol defining an operation or transformation	`=`, `+`, `*`, or `<`

Review: Operators & Literals

Quick: Yell it out!

- Literal
- Variable
- Operator

Symbols:

- `"hello_there"`
- `=`
- `"+"`
- `2`

Review: Operators & Literals

Quick: Yell it out!

- Literal
- Variable
- Operators

Symbols:

- `counter_strike`
- `/`
- `"3"`
- `spartan_name_117`

Review: f-strings

f-strings are string literals that have an `f` prefix.

This allows us to have `{}` inside the string that contains an expression. That expression will be evaluated into the string value.

```python
x = 100
y = 4.2
name = "Mark"
script_line = f"I had a dream where my GPA was a {y}!"
script_line = f"But really it is a  {y - 2}!"
script_line = f"There are {x} chickens outside..."
script_line = f"Oh, Hi there {name}."
```

Lecture Activity

L11:

What is printed?

```
x = 101
print("f\"{x}\"")
print(f"{x - 100}")
print(f"(x + 1) is equal to ({x} + {1})")
```

Types

In Python, we have different '`type`'s of variables!

The `type` of a variable determines how the value it has stored is interpreted and used.

This will become clearer as we use operators on these variables later in lecture.

Data Type	Purpose	Sample Values	Sample Operations
`int`	whole (integer) numbers	`3`, `-14`, `0`	`+`, `-`, `*`, `/`
`float`	numbers with fractional parts	`3.0`, `-14.32`, `0.0`	`+`, `-`, `*`, `/`
`str`	text	`"CIS 1100"`, `"False"`	`len()`, indexing & slicing

Strings

We have seen strings before, they contain a sequence of characters.

Even in our first program, we were using a string:

```
print("Hello World!")
my_string = "hey!"
print(my_string)
```

Using Operators on String Types

When using the `+` operator on Strings:

- the two strings are *appended together literally*.
- kept in the same order, from left to right.
- no spaces are inserted between them

```
example = "hello" + "world"
print(example) # prints "helloworld"
```

operator + with strings

+ is used as an operator to append strings together

It does not *overwrite* or modify any of the variables that are its *operands*.

Example:

```
x = "cis1100"
y = "com"
z = x + "." + y
print(x) # "cis1100"
print(y) # "com"
print(z) # "cis1100.com"
```

Reminder: The only way to change what a variable is equal to is with the = *operator.*

Calling functions "On" Strings

We can also call functions *on* strings to preform specific operations!

Consider this example:

```
x = "abcdefghijklmnop"
y = x.upper()
z = "boYmeEtsWoRld".lower()
w = "hey! What's up?".upper()
print(x)  # "abcdefghijklmnop"
print(y)  # "ABCDEFGHIKLMNOP"
print(z)  # "boymeetsworld"
print(w)  # "HEY! WHAT'S UP?"
```

Syntax: `<string>.func_name()`

Important: these only apply to cased characters, not punctuation/symbols!

We can use these functions 'on' anything that is a string!

More `string` functions

- `.upper()` makes a copy where all letters are uppercase
- `.lower()` makes a copy where all letters are lowercase
- `.capitalize()` makes a copy with its first character capitalized and the rest lowercased.
- `.strip()` makes a copy where all space before and after the characters are removed.
  - e.g. `"     hey!         "`.strip() becomes `"hey!"`.
- `str.replace(old, new)` makes a copy where all instances of the string `old` are replaced by `new`.
  - eg `"aaa".replace("a", "b")` becomes `"bbb"`.
- `a + b` makes a new string value that has the value of `b` attached to the end of `a`
  - *note: `a` and `b` are strings*

Common Mistakes with Variables

A couple of beginner mistakes to make when programming are:

- Forgetting that (most) operators do not modify variables that are operands
- improperly keeping track of values stored in variables

Let's take a look at a couple of problems!

Lecture Activity

What is printed at the end of the program?

S8:

```
initial_string = "   YOU are  all    ".strip()
initial_string = initial_string.lower()
corrected_string = initial_string.replace("you", "You")
corrected_string = corrected_string.replace("  ", " ")
emphasized_string = corrected_string + " AMAZING!!!"
final = emphasized_string.replace("!!!", "!")
print(final.capitalize())
```

Lecture Activity

What are the final value of all variables in this program?

C12:

```
neo = "the"
morpheus = "one"
matrix_code = f"{neo.upper() + morpheus.capitalize()}".replace("One", "Chosen")
morpheus = "Agents of the Matrix"
final_transformation = morpheus.replace("Agents", "Architects").lower()
neo = f"In {1999 + 24}, {matrix_code} rewrote: {final_transformation.capitalize()}"
oracle = f"{final_transformation}-{morpheus}"
```

Numerical Types

In python, we can store numbers in variables.

However, there is a distinction between two types:

- `int` These are Integers, meaning any positive or negative value (or zero).
  - e.g. `0`, `-3200`, `10`, `299792458`
- `float` These can store rational numbers and some special values
  - e.g. `3.14`, `8.3144`, `1.4142`, `2.718`, `infinity`, `-infinity`

Numerical Operators

- `+`: addition
  - x + y
- `-`: subtaction
  - x - y
- `/`: divide
  - x / y
- `*`: multiplication
  - x * y

Order of operations (PEMDAS) and evaluating from left to right still applies.

If you want to enforce what happens first or a specific order, use `(` and `)`.

Mixing Numerical Types

- If you use an operator on two `int`s you get an `int`
  - (except `/` then you get a `float`, why?)
  - *the motivation for this might not be clear yet!*
- If you use an operator on two `float`s, the result will be a `float`
- if you operate on an `int` and a `float` you get a `float` (why?)

Lecture Activity

What are the resulting types of the expressions and what will be printed?

S9:

```
x = 3 + 0.5 * 2
print(x)
```

S10:

```
x = (2 * 8) / 3
print(x)
```

More Assignment Operators

There are also a few more operators worth covering:

- `+=`

  ```
  x = "h"
  x += "i"
  // x here becomes "hi"
  ```

  This operator "adds" the two values together and sets the variable on the left equal to the result.
- Other variants: `-=`, `*=` and `/=` exist for numerical types (`*=` works on strings too!)

Other Arithmetic Operators

- `**` used for exponents.
  - e.g. 5 squared is written as `5 ** 2`
- `//` used for "integer division, rounds the result towards 0
  - `int // int` evaluates to an `int`
  - `3 // 2` evaluates to `1`
- `%` called "modulo" used to get the remainder of a division.
  - `5 % 2` evaluates to `1`
  - `9 % 3` evaluates to `0`

Lecture Activity

S10:

What does this evaluate to? `(10 % 3) ** 2 // 5`

If Time: Boolean Type

```python
x = True
y = False
print(x)
```

Comparison

A common way to get boolean values is through comparison.

- `==` checks if two things are equal
- `!=` checks if two things are NOT equal

`"Hello" == "hello"` evaluates to `False`

`5 != 3` evaluates to true

`"hi" == "hi"` evaluates to `True`

More on `bool` & a new type `None` next time

Reminder:

- Next lecture on Monday 01/27
- There is another check-in due before that lecture as well.
- Office Hours and Recitation start next week
  - Recitation attendance is counted, show up to your assigned recitation!
- HW00 is out and due Wednesday (1/29) at midnight