CIS 1100

Types & Variables! (Lecture)

Python
Fall 2024
University of Pennsylvania

Review: Variables

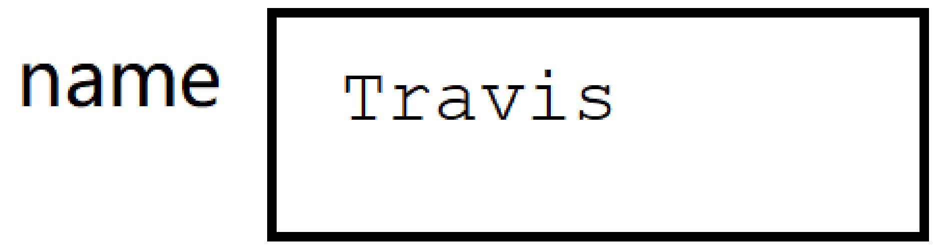
Review: Intro to Variables

Lets start with a simpler example:

```
name = "Travis"
```

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

You can think of a variable sort of like a box with a name attached to it. The value in the box can change over the course of the program, but can only hold one thing at a time.

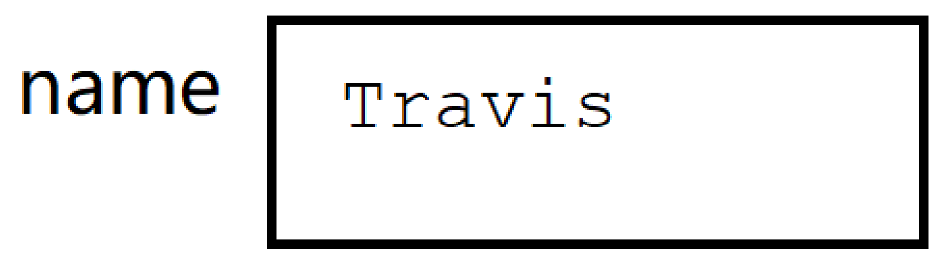


Review: Printing Variables

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

```
name = "Travis"
print(name)  # prints Travis
```

When we give the name of the variabled, it tells python to see what is stored "inside that box"

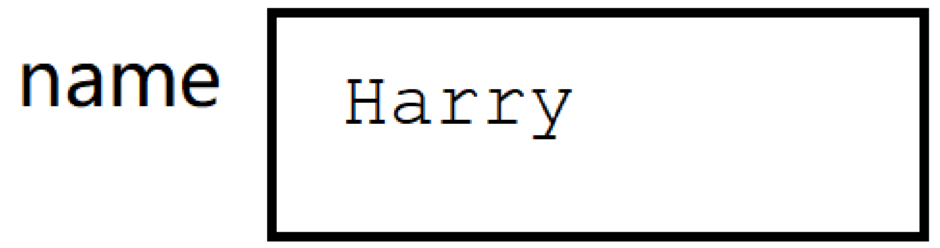


Review: Variables Change over Time

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

Consider:

```
name = "Harry"       <-----
name = "Travis"
print(name)
```

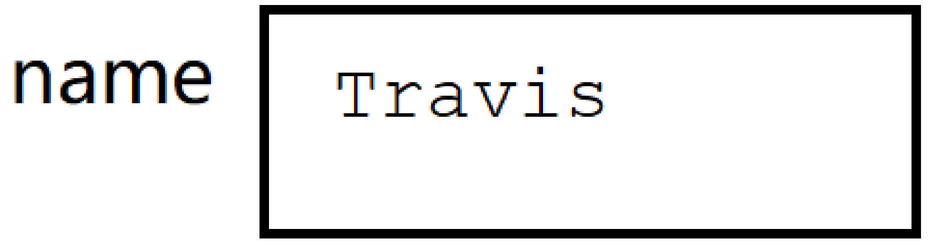


Review: Variables Change over Time

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

Consider:

```
name = "Harry"
name = "Travis"          <-----
print(name)
```



Review: Variable Naming

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

Consider the following varaibles, which ones are good variable names (and legal)

Use (M1) on your worksheet. Bubble in the corresponding
bubble if you think it is a good (and legal) variable name

a) `local_counter3`
b) `2s_compliment`
c) `HowdyPartner`
d) `where_It_Is`
e) `import`

Review: Expressions

- **Expressions** are portions of a program that have 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` or `x` or `last_name`
Operator	A symbol defining an operation or transformation	`=`, `+` or `*`

Review: Operators & Literals

Quick: Raise your fingers

- 1 Finger: Literal
- 2 Fingers: Variable
- 3 Fingers: Operators

Symbols:

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

Review: Operators & Literals

Quick: Raise your fingers

- 1 Finger: Literal
- 2 Fingers: Variable
- 3 Fingers: Operators

Symbols:

- `howdy`
- `+`
- `"3"`
- `title_fight`

Review: f-strings

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

Allows us to have `{}` inside the string that contain an expression. That expression will be evaluated into the string value.

Poll: What gets printed? (L11)

```
x = 3
print("{x}")
print(f"{x + 1}")
print(f"(x + 1) is equal to {x} + {1}")
```

Types

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

# there are others, but lets stick to these 3 for this lecture.

# More in videos & next lecture

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!")
```

operator `+` with strings

We can use the `+` operator to take two string values and append them together.

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

operator `+` with strings

Note: Using `+` on a varaible doesn't change the variable used in the `+`, it copies the value for the expression.

Example:

```
x = "title"
y = "fight"
z = x + " " + y
print(x) # "title"
print(y) # "fight"
print(z) # "title fight"
```

Calling functions "On" strings

We can also call functions *on* strings to preform some operation.

Consider this example:

```
x = "happy"
y = x.upper()
z = "World".lower()

print(x)  # "happy"
print(y)  # "HAPPY"
print(z)  # "world"
```

Syntax: `<string>.func_name()`

More string functions

- `.upper()` makes a copy where all letters are uppercase
- `.lower()` makes a copy where all letters are lowercase
- `.replace(old, new)` makes a copy of the string where all instances of `old` are replaced by `new`
- `a + b` makes a new string value that has the value of `b` attached to the end of `a`

What does this `z` evaluate to? (S7)

```
x = "you"
y = "ynrnrt 2".lower()
z = y.replace("nr", "e") + " " + x.upper()
```

Practice: Reassignment

Some easy mistakes to make when programming are:

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

lets practice these :)

Practice:

What are the final value of all variables in this program? (C12)

```
my = "ed"
ele = "ge"
name = f"{my + ele}".upper()
ele.uppper()
my = "Moons of Uranus"
x = my.replace("Uranus", "Neptune").lower()
my = "2015 feels like {2} years ago - {name}"
```

Numerical Types

Python can store numbers, but it makes a distinction between two types of numbers:

- `int` These are Integers, meaning any positive or negative value (or zero).
  - e.g. `0`, `-3200`, `10`
- `float` These can store rational numbers and some special values
  - e.g. `3.14`, `2.0`, `infinity`

Basic Operators

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

Order of operations (PEMDAS) read left -> right still applies.

If you are unsure about order of ops, just 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?)
- If you use an operator on two `float`s you get a `float`
- if you operate on an `int` and a `float` you get a `float` (why?)

What do these evaluate to and print?

- `3 + 0.5 * 2` (S8)
- `(2 * 8) / 3` (S9)

More Assignment Operators

There are also a few more operators worth covering:

- `+=` used to do both `+` and `=`

  ```
  x = "h"
  x += "i"
  ```

  is equivalent to

  ```
  x = "h"
  x = x + "i"
  ```

  works for numerical types as well

- Variants like `-=`, `*=` and `/=` exist for numerical types

Other Arithmetic Operators

- `**` used for exponents.
    - e.g. 5 squared is `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`

What does this evaluate to? `(10 % 3) ** 2 // 5` (S10)

If Time: Boolean Type

# if we don't get to it this lecture, then we will cover in next lecture

Another type that exists is `bool` which can represnt two values `True` or `False`

```
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 09/09
- There is another check-in due before that lecture as well.
- Office Hours and Recitation start next week
    - Recitation counts attendance, show up to your assigned recitation!
- HW00 is out and due Wednesday (9/11) at midnight
- Sunday Review Sessions start this weekend
    - every Sunday, 10-12am
    - check Ed for room announcement