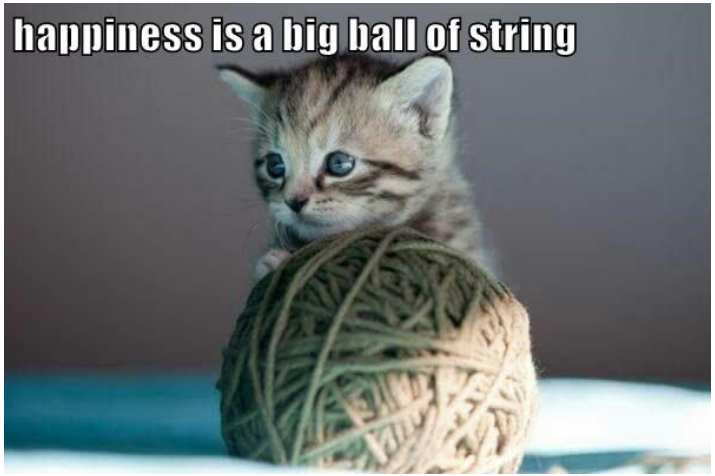
CS 1: Intro to CS

Python Functions and Strings



April 3rd, 2024

Today's Agenda

Identifying functions and data types in the real-world

- The function-as-machine model
  - Arguments (0 or more) and their data types (int, str, list, boolean, etc.)
  - Behavior (e.g. computing a total price from given cost and tax rate)
  - Output vs. return (e.g. printing a message, returning total price as a float)
- Pseudocode
  - Step-by-step instructions for the function's implementation ***after*** identifying arguments/return and ***before*** implementing code
- Intro to documentation
  - # for inline "source-code" comments
  - """..""" for function documentation (docstrings)

Collectively, an immersive introduction to what we'll be continuing to learn this term!

Mid-Week Study Tips

We'll be doing more live-coding/whiteboard activities today, but these slides are very useful reference and will be continued Friday (focused on variables, assignment, and scoping)

Post on Discord function/program ideas you come up with today using the format from class! An example is also posted in #fn-ideas, and be creative!

- For additional practice, try writing basic pseudocode and/or docstrings (either for your idea, or another student's)

Read the readings posted so far this week, attend lectures!

Try asking at least 1 question on Discord this week, whether it's specific to something El introduces or on #random; let's start the term off with active engagement :)

Make sure you know the difference between running a Python program vs. the >>> prompt (next slide)

Python So Far

Writing and running Python programs:

1. Running a program with the **`python3 <filename>`** command in the terminal
2. Writing and running Python in the Python interpreter (invoked in the terminal with just **`python3`**)

Our first **`print('Hello world!')`**

Getting started with the terminal to navigate your directories and run Python files

Command Line Recap

`$ pwd`

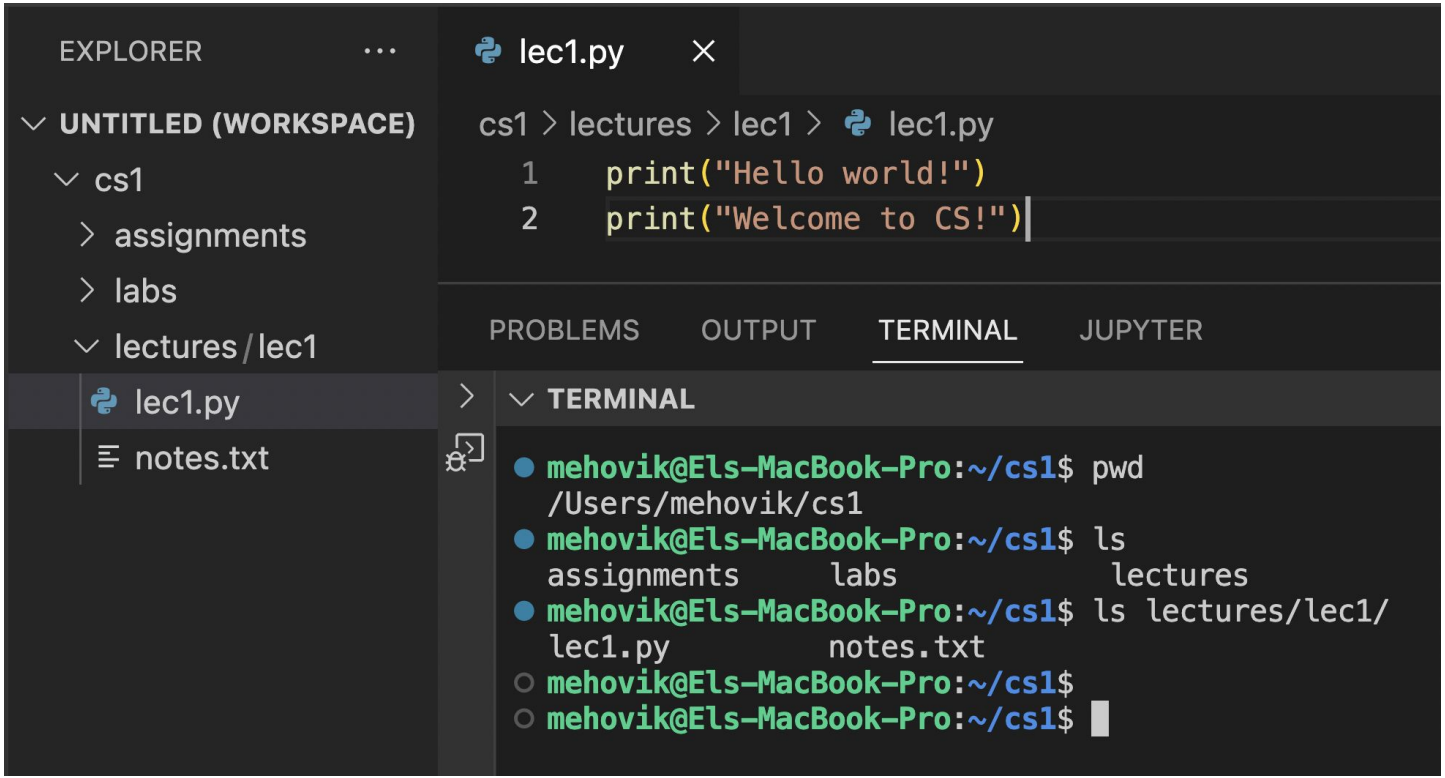
- "Print working directory"; outputs the system path for the current directory

`$ ls` or **`$ ls /path/to/subdirectory`**

- "list files/subdirectories" (if given path, lists those in the path location instead of current directory)

Note: The provided screenshots are from El's VSCode terminal as shown in Lecture 1; do not include **`$`** in your commands, this is the default for many (but not all) command line prompts

Command Line Recap



Command Line Recap

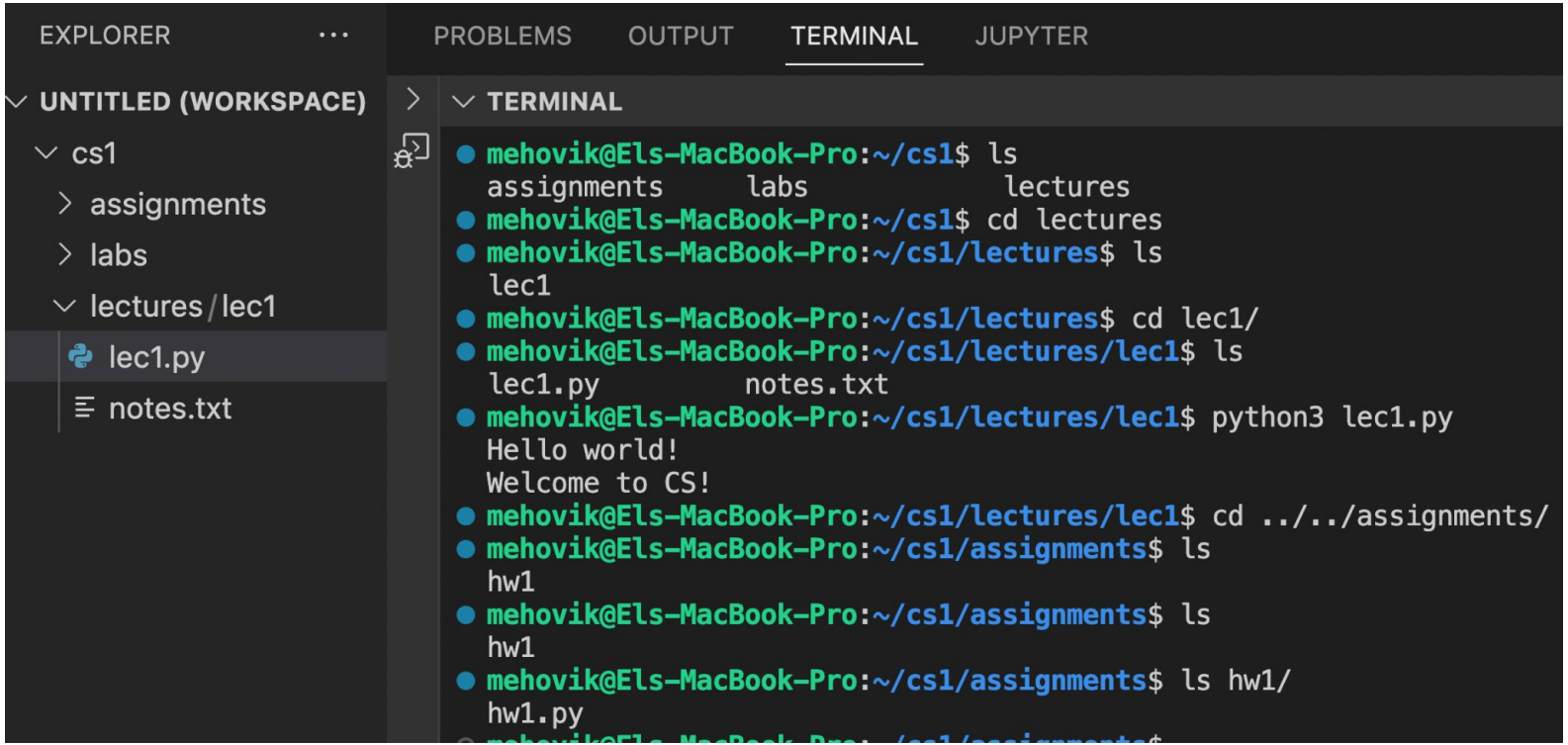
`$ cd <subdirpath>`

- "Change directory" to subdirectory path (see below)

`cd ..`

- "Go up one directory" (can compound to "go up and in")

Command Line Recap



Command Line Recap

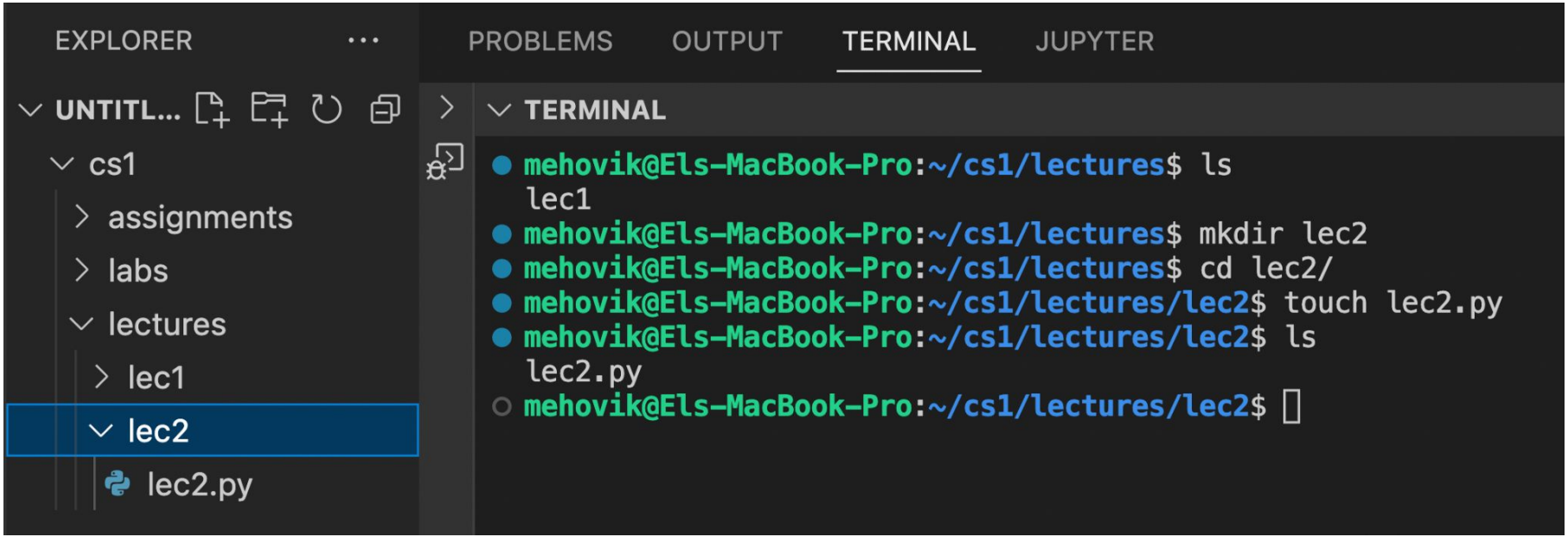
`$ touch <filename.ext>`

- Create a new *file* called **`<filename.ext>`** (e.g. **`touch lec2.py`**)

`$ mkdir <dirname>`

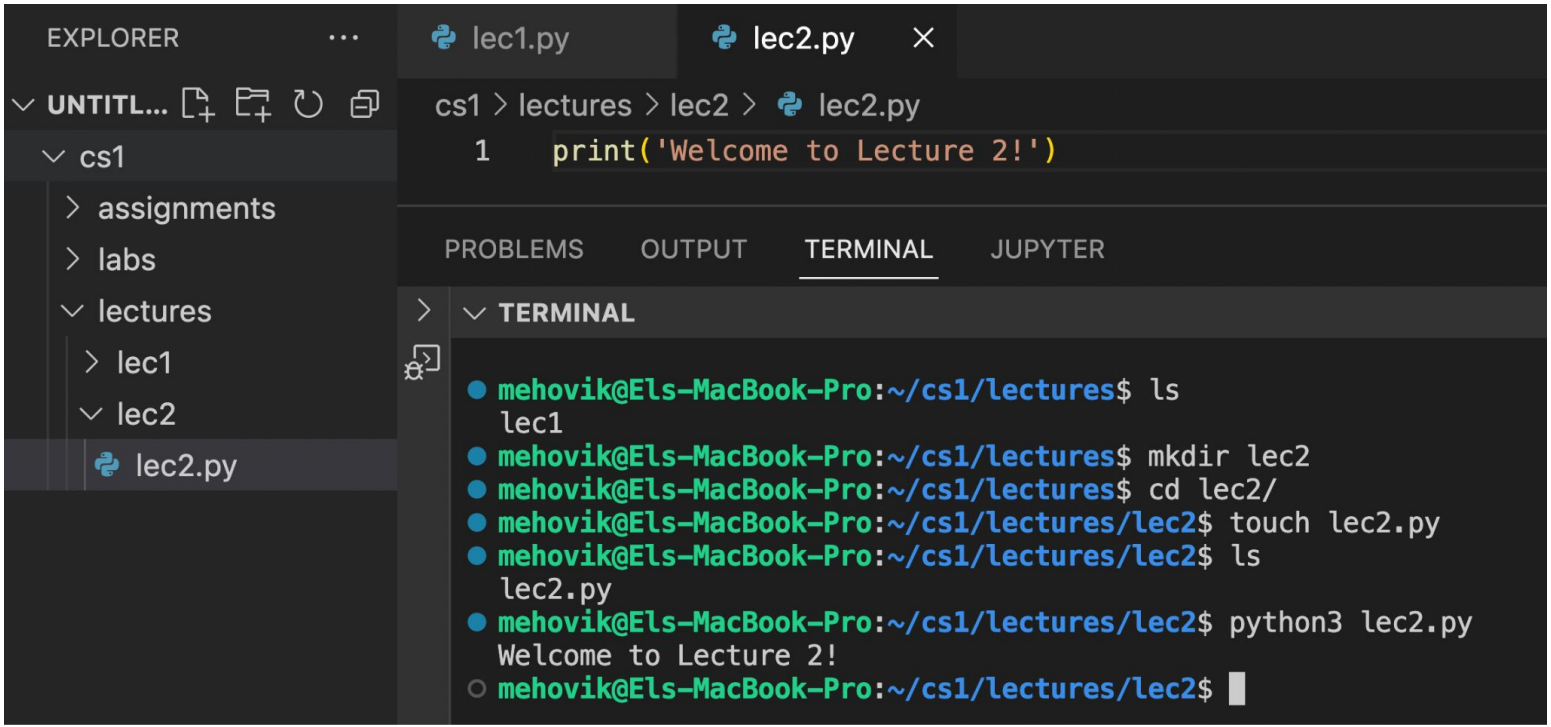
- Create a new *directory* called **`<dirname>`**
  - (e.g. **`mkdir lec-practice`**)

Command Line Recap



Note: You can also use the "new file" and "new folder" icons on the left Explorer pane if you prefer

Command Line Recap



Command Line Shortcuts

There are a few shortcuts you can use in the command line, but the ones you'll see El using often are:

1. <Tab> to autocomplete a file path (e.g. **`cd le<Tab>`** -> **`cd lectures/`**)
2. The Up arrow to go up to the last command

Today's Learning Objectives

Introduce **variables** and **assignment** to keep track of data

Introduce useful built-in Python **functions**

Learn why, how, and when to "package" code into *our own* reusable functions

Continued Friday:

Understand variable **scoping** with functions

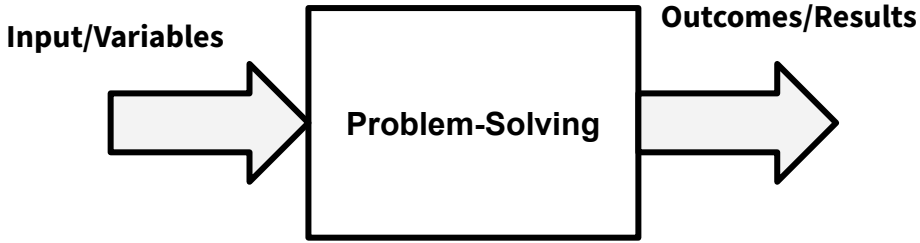
Understand how to use **strings** and common pitfalls

Learn how to use string formatting with the `format` method

What is a Program, Really?

Even if you do not have any programming experience, you have all solved problems in the real-world:

- Math problems
- Chemistry/Biology/Physics problems
- Budgeting
- Planning your schedule for this term
- Applying to Caltech
- Prioritizing your commitments/obligations
- Making arguments/finding compromises with people
- Finding the best deal for a new computer
- Solving puzzles/strategizing in board or video games
- …



Fundamentals of Programming

Programming is all about formalizing problem-solving using a language of choice (we happen to use Python in CS 1)

This week, we'll introduce the fundamentals of programming to solve a variety of problems:

- Arithmetic and expressions
- Variables and assignments
- Datatypes
- Functions
- Scope
- Program Decomposition

Activity

In Discord #lecture, share your response to the following question:

What is an example real-world problem you could model as a function of input to output?

Some examples:

- Given a temperature in Fahrenheit, convert to Celsius
- Given a unit in feet (ft), convert to meters (m)
- Given a birthday, determine the age in years
- Given a Pokemon type, determine its weakness
- Given a favorite music genre, provide 10 recommended Spotify songs
- …
-

Previous Student Ideas

Given dna output the corresponding proteins (this is an upcoming Mini Project!)

Given cost of food at a restaurant, determine tip

Given food allergies, what items on the menu can you eat

Given a Pokemon type, design a six-membered team such that the maximum "type coverage" is achieved (aka design a gym leader or elite four team).

Student Ideas

Given a date, return the day of the week.

```
mehovik@Els-MacBook-Pro:~/cs1/lectures/lec2$ date
Fri Sep 30 13:46:08 PDT 2022
mehovik@Els-MacBook-Pro:~/cs1/lectures/lec2$ date +%u
5
mehovik@Els-MacBook-Pro:~/cs1/lectures/lec2$ date +'%A'
Friday
```

^ There is a bash command for this!

Student Ideas

Given a date, return the day of the week.

```
>>> from datetime import datetime
>>> # ^ we will learn more about "libraries" soon!
>>> datetime.today()
datetime.datetime(2022, 9, 30, 13, 50, 6, 574985)
>>> datetime.today().strftime('%u')
'5'
>>> datetime.today().strftime('%A')
'Friday'
>>>
```

You can also do this in Python!

Arithmetic and Expressions

Arithmetic expressions contain numbers (operands) combined with symbols (operators) which compute values given the numbers

Operators: + - * / etc.

Numbers can be integers (no decimal point) or floating-point (with decimals)

- Floating-point is an approximation to real numbers

Operator Precedence

What does `1 + 2 * 3` mean?

It could mean

- `1 + (2 * 3)`
- `(1 + 2) * 3`

Computer languages have precedence rules to determine meaning of ambiguous cases

Operator Precedence

What does `1 + 2 * 3` mean?

It could mean

- `1 + (2 * 3)` Correct!
- `(1 + 2) * 3`

Computer languages have precedence rules to determine meaning of ambiguous cases

Here, * has higher precedence than +, so the first meaning is correct

Operator Precedence

In general, + and - have lower precedence than * and /

The ** (exponentiation) operator is even higher precedence than * and /

```
>>> 2 * 3 ** 4
```

```
162
```

Use parentheses to force a different order of evaluation if you need it

```
>>> (2 * 3) ** 4
```

```
1296
```

Variables and Assignment

Often, we want to give names to quantities

In Python, use the = (assignment) operator to do this:

```
>>> salary = 18.5
```

From here on, `salary` stands for `18.5`

```
>>> salary * 20
```

```
370
```

Variables and Assignment

Names assigned to can be reassigned:

```
>>> salary = 18.5
>>> salary
18.5
>>> salary = 30
>>> salary
30
```

Variables and Assignment

Names of variables ("identifiers") can only consist of the letters a-z, A-Z, the digits 0-9, and the underscore (_)

Identifiers also cannot start with a digit (avoids confusion with numbers)

Identifiers can't contain spaces!

Note: Case of letters is significant

- Foo is a different identifier than foo

```
a = 10

b1 = 20

this_is_a_name = 30

&*%$2foo? = 40 # not valid!
```

Variables and Assignment

Can have expressions on the right-hand side of assignment statements:

```
>>> salary = 18.5

>>> weekly_salary = salary * 20

>>> weekly_salary

370
```

The expression is terminated by the end of the line

Variables and Assignment

Can use results of previous assignments in subsequent ones:

```
>>> x = 15
>>> y = x * 5
>>> y
75
>>> z = x + y
>>> z
90
>>> z = z + 10
>>> z
100
```

Variables and Assignment

Evaluation rule for assignment statements:

1. Evaluate the right-hand side
2. Assign the resulting value to the variable on the left-hand side

This explains why `z = z + 10` works:

- previously, z was 90
- evaluate `z + 10` to 100
- assign 100 to z (new value)

Variables can vary!

Types

Data in programming languages is subdivided into different "types":

- integers: `0 -43 1001`
- floating-point numbers: `3.1415 2.718`
- boolean values: `True False`
- strings: `'foobar' 'hello, world!'`
- and many others

Types

In Python, the same variable can hold data of different types at different times:

```
>>> a = 'foobar'
>>> a
'foobar'
>>> a = 3.1415926
>>> a
3.1415926
```

What might be an issue with this?

Comments

Comments are lines in the source code that are notes to the reader(s), while Python just ignores them

Comments start with # and continue to the end of the line

```
# U.S. dollars per hour
salary = 18.5 # everything after the comment symbol is ignored
```

We'll learn other ways to document your code properly next week

Functions

A function takes some input data and transforms it into output data

Functions must be defined and then called with the appropriate arguments

A few functions are built-in to Python so we don't have to define them ourselves:

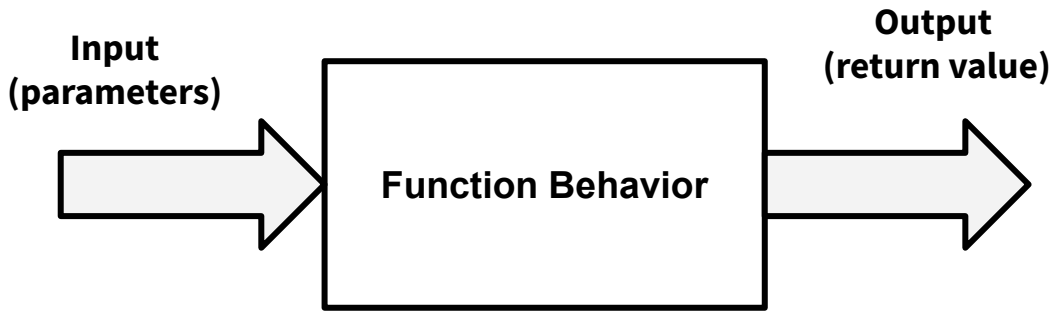
- `print(x)`
- `input(x)`
- `type(x)`
- `int(x), float(x), str(x)`
- `min(x, y, ...), max(x, y, ...)`
- `help(), help(fn)`
- `...`

Anatomy of a Function

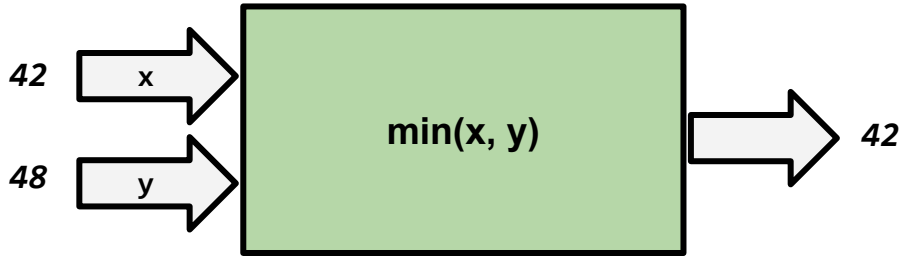
A function is like a machine to perform tasks and possibly return some result

Every function has:

- Behavior (body)
- Parameters (optional)
- Return value (optional)



Example with built-in `min` function:



Defining and Calling Functions

Functions may have parameters passed to help generalize functionality and may also specify a return value with the return keyword (**None** if no return specified)

Definition Syntax:

```
def name(<parameters>):
    <body>
    return <value> # optional
```

Definition Examples:

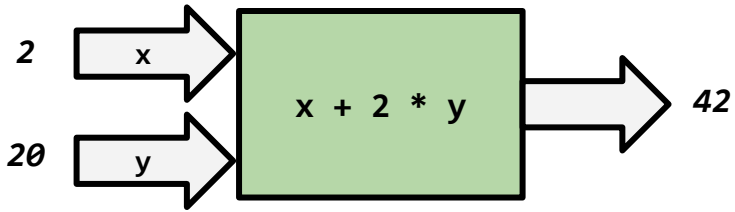
```
def say_hello(name):
    print('Hello ' + name + '!')

def f(x, y):
    return x + 2 * y
```

Function Call Examples:

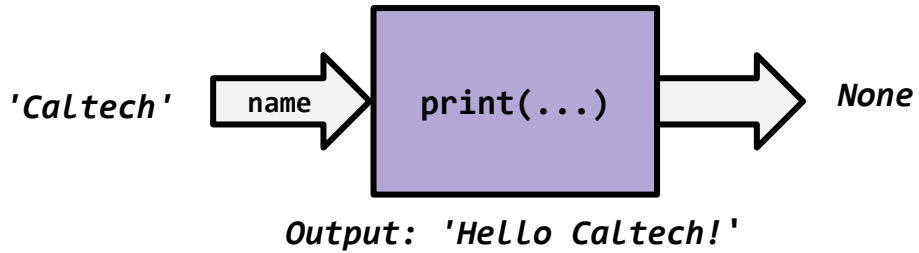
```
say_hello('world')   # Hello world!
say_hello('Caltech') # Hello Caltech!
ans = f(2, 20)       # ans == 42
```

Functions as Machines

```
1 # Defining the function
2 def f(x, y):
3     return x + 2 * y
4
5 # Calling the function
6 ans = f(2, 20)
```




```
1 # Defining the function
2 def say_hello(name):
3    print('Hello', name, '!')
4
5 # Calling the function
6 say_hello('Caltech')
```



Scope is Important!

So far, our variables have been defined top-down - later assignments shadow will shadow earlier ones.

Functions introduce their own **local** scope - **variables inside functions only exist in during the lifetime of a function call**.

```
1  def f(x, y)
2      return x + 2 * y
3
4  ans1 = f(2, 20) # 42
5  ans2 = f(x, 20) # error! x is not in scope here
```

Parameters vs. Arguments

Formal parameters are simply names for the argument values passed in a function call. The **position of arguments** will determine what formal parameter name they are assigned.

They have no relationship to other variable names in the program and will override other variables if there is a naming conflict.

```
1  def f(x, y)
2      return x + 2 * y
3
4  a = 2
5  b = 20
6  ans1 = f(a, b)
7  ans2 = f(b, a) # b and a are mapped to x and y in f, respectively
```

Practice

What is the result of executing the following program? (PythonTutor demo)

```
1  x = 1
2  y = 2
3  z = 3
4
5  def square(x):
6      return x * x
7
8  def mystery(x, y, z):
9      print("x: ", x, "y: ", y, "z: ", z)
10
11 mystery(x, y, z)
12 mystery(x + y, x, square(y))
```

Practice

What is the result of executing the following program? (PythonTutor demo)

```
1  x = 1
2  y = 2
3  z = 3
4
5  def square(x):
6      return x * x
7
8  def mystery(x, y, z):
9      print("x: ", x, "y: ", y, "z: ", z)
10
11 mystery(x, y, z)
12 mystery(x + y, x, square(y))
```

```
Output:
x: 1 y: 2 z: 3
x: 3 y: 1 z: 4
```

Next Time

Implementing our own functions in Python

Program and function scope

Using the Python debugger to walk through a program execution with function calls

More Strings and String formatting methods

Week 1 Action Items

Make sure to double-check your (Caltech) email for a CodePost invite sent yesterday (check your spam if you don't see it in your inbox)

- If you find it and activate it using the link, but still have issues logging in, try closing your browser and logging in again via a new browser session

Read the first three readings to review this week's material and preview Friday's lecture material

HW 1 will be posted by Friday, due next Thursday at 11:30PM via CodePost