CS10 NEWS

- Do iclicker attendance
- Dan's OH Friday in 606 Soda to review Midterm
- Midterm 2 this weekend (remember you only have to do the questions you have not yet aced!)

UC Berkeley Teaching Professor Dan Garcia

The Beauty and Joy of Computing

Python II – Built-in Types

Favorite / Hated Programming Languages

StackOverflow listed the most hated and loved programming languages. In order of the most hated: Perl, Delphi, VBA, PHP, Objective-C, Coffeescript, Ruby, C, Java and C++. Least disliked languages: R, Python, Typescript, Go, and Rust

www.techrepublic.com/article/the-10-programming-languages-developers-hate-the-most/

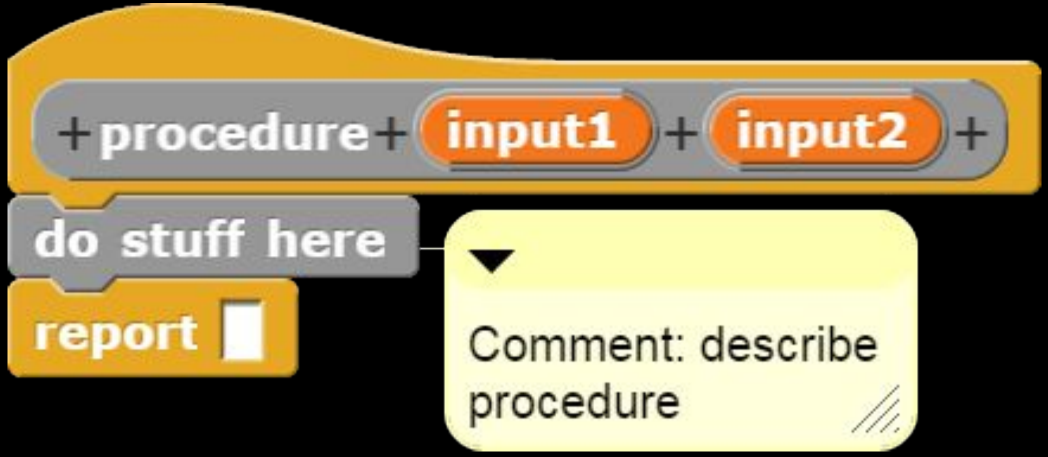
python™

Potpourri

Python Programming Modes

- Interactive Mode
  - Type `python3` on the command line
  - `python3 -i <filename>`
  - Executes commands in real-time as you type them
  - Good for trying things out
- Normal Mode
  - Write programs in files with extension `.py` (text editor)
  - Execute scripts all at once
    - On command line: `python3 <filename>`
    - Must be in same folder as file!
  - Good for writing functions and programs

Procedures



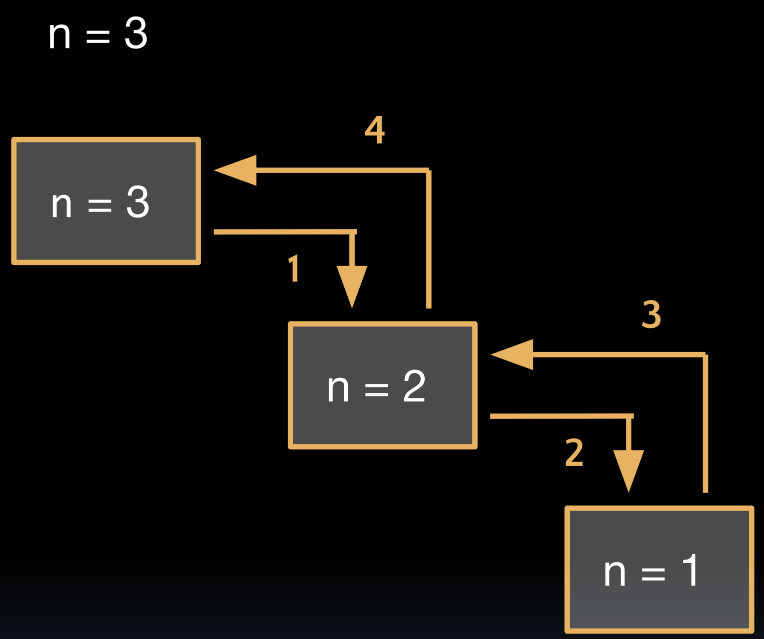
```python
def procedure(in1, in2):
    # comment here
    do_stuff_here()
    return <expression>
```

- Key differences
  - No distinction of procedure type
  - Must use parentheses for parameters
  - No spaces in variable or procedure names!
  - `return` instead of report
  - Indentation is VERY important in Python

Variable Scope

- These still apply just like in Snap*!*

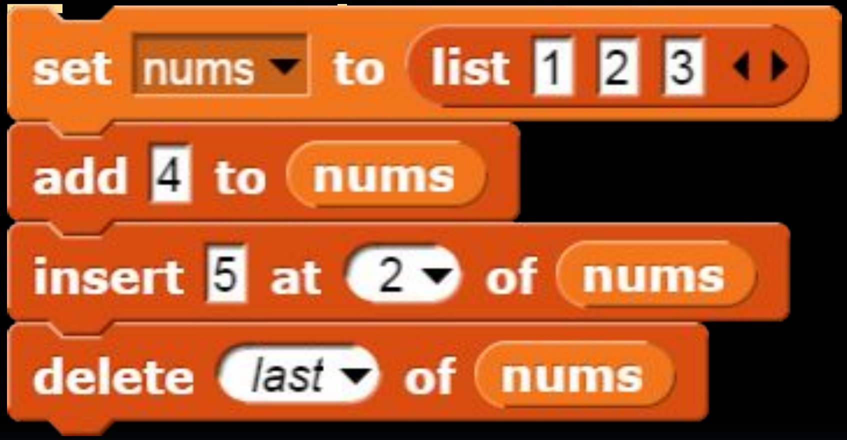
```
def fact (n):
    # recursive factorial
    if n < 2:
        return 1
    else:
        return n * fact(n-1)

n = 3
print('fact(',n,') =',fact(n))
```



- Un-indented **n** is global
- **n** within function is local to it

Object Functions

- In Python, many data types and data structures have built-in functions
  - Access via “dot” notation. List



```
nums = [1, 2, 3]
nums.append(4)
nums.insert(1,5)
nums.pop()
```

- So how do you know what exists?
  - `docs.python.org/3/tutorial/datastructures.html`
  - Python’s `dir` function (e.g. `dir(nums)`)

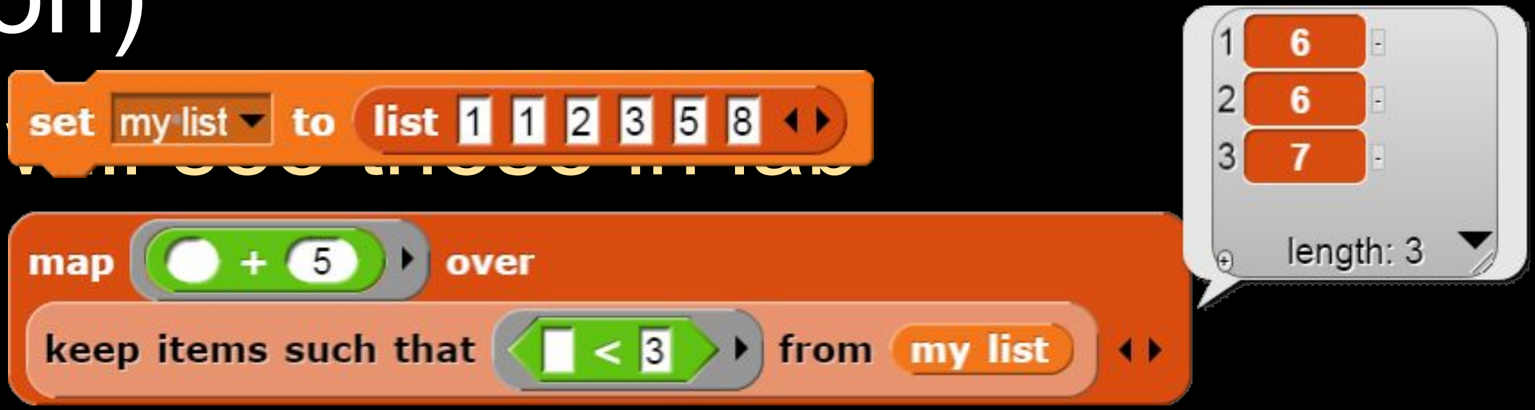
APIs

- Application Programming Interface (API)
  - APIs allow for “black box” use of pre-programmed elements
  - Provide abstraction and save work
- Most often accomplished in Python using `import` command
  - Import local files (same folder) using file name (no `.py` extension)
  - Import built-in Python modules

List Comprehensions

Map and Filter

- The direct equivalent of **`map`** and **`keep`** (Snap*!*) are **`map`** and **`filter`** (Python)
  - You will see these in lab



set my list to list 1 1 2 3 5 8

map (+ 5) over keep items such that (< 3) from my list

1	6
2	6
3	7

length: 3

```
def plus_5(x):
    return x + 5
def less_than_3(x):
    return x < 3

ans = map(plus_5, filter(less_than_3, [1, 1, 2, 3, 5, 8]))
```

Anonymous Functions

- Anonymous functions in Python are called lambda functions
  - Defined by: `f = lambda x,y: x + y`
  - Don’t need special run/call, just treat variable as functi `f(2, 3)` e:

```
ans = map(lambda x: x + 5,
          filter(lambda y: y < 3, [1, 1, 2, 3, 5, 8]))
```

List Comprehensions

- List comprehensions are a concise way to create lists
  - Enclosed in brackets [ ] much like a list is
  - Uses keywords **`for`**, **`in`**, and **`if`** in somewhat intuitive ways to try to make it more human-readable
  - Takes practice to get used to; you'll see in lab

```python
ans = [x + 5 for x in [1, 1, 2, 3, 5, 8] if x < 3]
```

Idiomatic Python (a.k.a Pythonic)

- While equivalent, most Python programmers use list comprehensions instead of HOFs
  - No direct translation of list comprehension in Snap*!*
- Idiom: “a style or form of expression that is characteristic of a particular person, type of art, etc.”
  - Idiomatic Python is code specific to Python
  - It is also referred to as ‘Pythonic’

Sequences

Iterables

- An iterable is an object capable of returning its members one at a time
  - In Snap*!*, only lists are iterable
    - E.g. item-#-of-list
  - Python has many kinds
- Sequences
  - Strings, Lists, Tuples, Ranges
- Sets and Dictionaries

Strings

- Three ways to specify in Python:
  - Single or double quotes
  - Triple double quotes preserves formatting

```
s1 = 'hello bob'
s2 = "hello alice"
s3 = """this is
  a string on
  many lines"""
```

- Access with bracket notation
  - Just like a list

```
print(s2[1:4])
```

- Strings are *immutable* objects

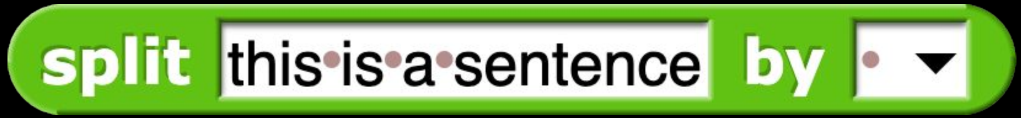
```
s2[0] = 'y'
```

  - `TypeError: 'str' object does not support item assignment`

String Functions

- Many handy String functions exist!
  - See online or type `help(str)` in interactive Python

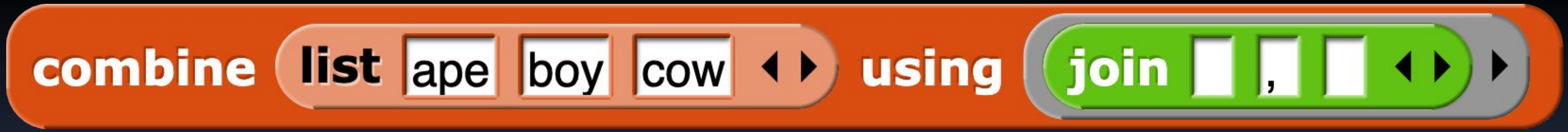
```
"this is a sentence".split()
```



```
"blown to bits".title()
```

$\longrightarrow$ “Blown To Bits”

```
', '.join(['ape','boy','cow'])
```



```
len('ultimate')
```

Tuples

- Tuples are almost exactly like lists, except:
  - Use parentheses `()` instead of brackets `[]`
  - They are *immutable*

```
some_tuple = (1, 5, 10, 4, 7, 16, 2)
some_list  = [1, 5, 10, 4, 7, 16, 2]
```

- What good is a “less powerful” list?
  - Prevents you from accidentally mutating when you don’t intend to
  - They can be used as keys to dictionaries

Ranges

- Produces regularly-spaced numbers
  - Form:
    ```
    range(end)
    range(start, end)
    range(start, end, step)
    ```
  - **`end`** value is *not* included
  - An immutable sequence
- Example:
  ```
  total = 0
  for x in range(4):
      total = total + x

  print(total)
  ```
  $\rightarrow$ 6

Using Sequences

- As you might expect, iterables are most useful with iterators
  - Looping – **`for`** loop takes an iterable
  - Examples:

```
for i in range(1,10,2):
for i in (3,9,2,4):
for state in ['AL','AK','AZ','AR']:
for char in "BJC":
```

- Also generally good for storing data
  - Easy access via indexing

Type Conversion (a.k.a Casting)

- What if you *did* want to mutate an immutable object?
  - Need to convert first
- Python includes lots of conversion functions
  - When in doubt, just try the object name

```
list(range(1,10,2))
print('n = ' + str(n))
print(4 + int("6"))
```

- Once converted, can use object functions of new object type

Sets and Dictionaries

Sets

- Sets are similar to lists/tuples except they are NOT sequences
  - Can't use bracket notation `[]` to access items
- Sets are unordered collections of *distinct* objects
  - Created using curly braces `{}`
  - Can convert to using `set()` function on ANY sequence
  - Supports set operations (union, intersection, difference,

```python
set1 = {1, 3, 5}
set2 = set(range(3,8))
set1 = set1.union(set2)
print(set1)
```

```
$ python set_def.py
{1, 3, 4, 5, 6, 7}
```

Example: Counting Unique Items

- One common trick using sets
  - See the link above if you're curious about this sentence:

```python
words = "Buffalo buffalo Buffalo buffalo buffalo
buffalo Buffalo buffalo".split()
unique_words = set(words)

print(unique_words)
print(len(words), '->', len(unique_words))
```

```
$ python basic_sets.py
{'buffalo', 'Buffalo'}
8 -> 2
```

Dictionaries

- More powerful list – instead of numerical index, store/access w/key (ANY immutable value)
  - Also created using curly braces { }, but entries look different
  - Incredibly

```
directory = {'Dan': '777 Soda'}
directory["Josh"] = "779 Soda"

print(directory)
print(directory["Dan"])
print(directory["josh"])
```

```
{'Josh': '779 Soda', 'Dan': '777 Soda'}
777 Soda
KeyError: 'josh'
```

Dictionary Functions

- Some handy dictionary functions:
  - **`.keys()`** returns a sequence of dictionary's keys
    - In no particular order!
  - **`.values()`** returns a sequence dictionary's values
    - In no particular order!
  - **`key in dictionary`** checks if key is in the

```
numerals = {'I': 1, 'V': 5, 'X': 10}
print(list(numerals.keys()))
print(list(numerals.values()))
print('L' in numerals)
```

```
['I', 'X', 'V']
[1, 10, 5]
False
```

L18 Select the true statement

With large projects, it's best to put all your code in a single Python file

To define a global variable in Python, you must use capital letters

Tuples are just like lists, except Tuples are mutable

Sets may have duplicate entries

A key may have more than one value in the dictionary

A value may have more than one key in the dictionary

Summary

- Concepts we learned in Snap*!* apply to Python
  - Ex: Variable Scope, Mutability, Anonymous Functions
- Python has many built-in object functions
  - Often need to look them up online
  - Sometimes necessary to convert between object types
- List comprehensions are unique to Python
  - Offer human-readable way to implement HOFs
- Sequences good for easy access via indexing
  - Lists, Strings, Tuples, Ranges
- Sets and Dictionaries offer more complex ways to access and manipulate collections of objects