F is for ...

- Function
  - Key to all programming
- Floating Point
  - Decimal numbers aka Python float
- File
  - Sequence of stored bits

Annie Easley

- American computer scientist, mathematician, and rocket scientist
- Worked at NACA and NASA
- BS in Math, Cleveland State
- Leader in developing the software for the Centaur rocket stage

*On microaggressions: “If I can’t work with you, I will work around you”*
Learning Goals: Faces

• Understand differences and similarities:
  • Function definitions vs function calls
  • Functions with return statements vs those without
  • Functions with parameters vs those without
  Functions can be arguments

• Be creative and learn lesson(s) about software design and engineering
  • Create a small, working program, make incremental improvements.
  • Read the directions and understand specifications!

Name vs Value vs Type

- Functions as Parameters
- Debugging
- List concatenation and nesting
- Mutability
What are the arrows?

- **Name:** Enzo’s Pizza Co.
- **Address (arrow):** 2608 Erwin Rd # 140, Durham, NC 27705
- **Value:** Physical Store

Functions can be arguments

```
1 def enzospizzaco():
2     print("Pizza!")
3     return "2608 Erwin Rd # 140, Durham, NC 27705"

5 def eatfood(where):
6     print("Let's go eat!")
7     address = where()
8     print("The address is", address)
9     if __name__ == '__main__':
10        eatfood(enzospizzaco)
```

Pass the address of the function
Pizza2.py - Pass multiple functions to eatfood

```python
7 def naanstop():
8     print("Indian cuisine!")
9     return "2812 Erwin Road, Durham, NC 27705"
10
11 def enzospizzaco():
12     print("Pizza!")
13     return "2608 Erwin Rd # 140, Durham, NC 27705"
14
15 def eatfood(where):
16     print("Let's go eat!")
17     address = where()
18     print("The address is", address)
19
20 if __name__ == '__main__':
21     eatfood(enzospizzaco)
22     eatfood(naanstop)
```

Output of Pizza2.py

```
C:\Users\Susan\AppData\Local\Programs\Python\Python36\python.exe pizza2

Let's go eat!
Pizza!
The address is 2608 Erwin Rd # 140, Durham, NC 27705
Let's go eat!
Indian cuisine!
The address is 2812 Erwin Road, Durham, NC 27705

Process finished with exit code 0
```

In Assignment 1 Faces

```python
# def face_with_mouthAndEyes(mouthfunc, eyefunc):
    print(part_hair_squiggly())
    print(eyefunc())
    print(part_nose_up())
    print(mouthfunc())
    print(part_chin_simple())
```

In Assignment 1 Faces

```python
# def face_with_mouthAndEyes(mouthfunc, eyefunc):
    print(part_hair_squiggly())
    print(eyefunc())
    print(part_nose_up())
    print(mouthfunc())
    print(part_chin_simple())
```

Two parameters that are functions!

Name of function, no parentheses

Add parentheses when ready to call function
In Assignment 1 Faces

```python
def face_random():
    eyefunc = part_eyes_sideways
    x = random.randint(1, 3)
    if x == 1:
        eyefunc = part_eyesAhead

    # now call the function
    face_with_mouthAndEyes(mouthfunc, eyefunc)
```

Variable whose values is a function name
Change variable's value
Pass function as arguments

What is the code missing?
Finish if statement to have three choices for eyes
Similar code for mouth choices
How Not To Debug

• Bad (but tempting) way to debug
  • Change a thing. Does it work now?
  • No ... another change ... how about this?
• Trust doctor if they say?
  • “Ok try this medicine and see what happens?”
• Trust mechanic if they say?
  • “Let’s replace this thing and see what happens”

It may be easy, but that doesn’t make it a good idea!

Debugging Steps

1. Write down exactly what is happening
   1. input, output, what should be output
   2. ____ happened, but ____ should happen
2. Brainstorm possible reasons this is happening
   1. Write down list of ideas
3. Go through list
4. Found it?
   1. Yes, fix it using the 7-steps
   2. No, go back to step 2

• Finding what is wrong + fixing it
  • Finding is its own skill set, and many find difficult
  • Fixing: revisit Step 1—5
Debugging Steps

1. **Write down exactly what is happening**
   1. input, output, what should be output
   2. _____ happened, but _____ should happen
2. **Brainstorm possible reasons this is happening**
   1. Write down list of ideas
3. **Go through list**
4. **Found it?**
   1. Yes, fix it using the 7-steps
   2. No, go back to step 2

---

**Relate W’s to Debugging**

- **Who was involved?**
- **What happened?**
- **Where did it take place?**
- **When did it take place?**
- **Why/How did it happen?**

---

**Translate these questions to debugging**

- **Who was involved?**
- Which variables are involved?
- **What happened?**
- What kind of error/bug is it?
- **Where did it take place?**
- Where in the code did this happen?
- **When did it take place?**
- Does it happen every time? For certain cases?
- **Why/How did it happen?**
- Given the answers to the above, how did the error/bug happen?
Step 7 -> Steps 1-4 or 5

Which year is a leap year?

- A Leap Year must be divisible by four.
- But Leap Years don't happen every four years ... there is an exception.
  - If the year is also divisible by 100, it is not a Leap Year unless it is also divisible by 400.

WOTO-2: Buggy Leap Year

WOTO-2: Buggy Leap Year

```python
7     def is_leap_year(year):
8         if year % 4 == 0:
9             return True
10        if year % 100 == 0:
11            return False
12        if year % 400 == 0:
13            return True
14        return False
```

Input: 1900
Output: True
Should be: False
WOTO-2: Buggy Leap Year

• Who? (Which variables)
  year

• What kind of bug is it?
  Semantic error

• Where in the code?
  One of the places it returns True

• When does it happen?
  Input: 1900, but not 2016 nor 2019

• Why/How did it happen?
  A property 1900 has but not 2016 and 2019

Buggy Leap Year – add print tests

```python
def is_leap_year(year):
    if year % 4 == 0:
        return True
    if year % 100 == 0:
        return False
    if year % 400 == 0:
        return True
    return False

# Additional print tests
print('Is 2016 a leap year? (should be True)', is_leap_year(2016))
print('Is 2019 a leap year? (should be False)', is_leap_year(2019))
print('Is 1900 a leap year? (should be False)', is_leap_year(1900))
```

Output:

WOTO-2: Buggy Leap Year – Which “return true”?

```python
def is_leap_year(year):
    if year % 4 == 0:
        return True
    if year % 100 == 0:
        return False
    if year % 400 == 0:
        return True
    return False

# Additional print tests
if __name__ == '__main__':
    print('Is 2016 a leap year? (should be True):', is_leap_year(2016))
    print('Is 2019 a leap year? (should be False):', is_leap_year(2019))
    print('Is 1900 a leap year? (should be False):', is_leap_year(1900))
```

Output:
Buggy Leap Year – Which “return true”? 

```python
7 def is_leap_year(year):
8    if year % 4 == 0:
9        print("DEBUG: if year % 4 == 0:")
10       return True
11    if year % 100 == 0:
12       return False
13    if year % 400 == 0:
14       print("DEBUG: if year % 400 == 0:")
15       return True
16    return False
```

Output:
```
DEBUG: if year % 4 == 0:
```

Add prints to figure out which return True for 1900

Add prints to figure out which return True for 1900

Correct Leap Year – ifs correct order

```python
7 def is_leap_year(year):
8    if year % 400 == 0:
9        return True
10    if year % 100 == 0:
11        return False
12    if year % 4 == 0:
13        return True
14    return False
```

Output:
```
Is 2016 a leap year? (should be True) True ✓
Is 2019 a leap year? (should be False) False ✓
Is 1900 a leap year? (should be False) False ✓
```

Buggy Leap Year – Which “return true”? 

```python
7 def is_leap_year(year):
8    if year % 4 == 0:
9        print("DEBUG: if year % 4 == 0:")
10       return True
11    if year % 100 == 0:
12       return False
13    if year % 400 == 0:
14       print("DEBUG: if year % 400 == 0:")
15       return True
16    return False
```

Output:
```
DEBUG: if year % 4 == 0:
Is 2016 a leap year? (should be True) True
Is 2019 a leap year? (should be False) False
DEBUG: if year % 400 == 0:
Is 1900 a leap year? (should be False) True
```

This True returned!

Why Leap Year Buggy?

- Why: Should not always return True if year is divisible by 4
- Solution: Check first for %400, then %100, and finally %4

Steps 4-7: Devise Algorithm → Translate to Code → Test Program

Implementation Problem: Program is Incorrect
List Concatenation

- String concatenation:
  - "hi" + " there" == "hi there"

- List concatenation:
  - [1, 2] + [3, 4] == [1, 2, 3, 4]

List examples

- [1, 2 ] + [ 3, 4]
- lst1 = ['a', 'b']
- lst2 = [5, 6]
- lst1 + lst2
- lst1 + “c”
- lst1 + [“c”]

Nested Lists

- Lists are heterogenous, therefore!
  - lst = [1, ‘a’, [2, ‘b’]] is valid
  - len(lst) ==

- How to index?
  - [...] all the way down
Nested Lists

- Lists are heterogenous, therefore!
  - lst = [1, ‘a’, [2, ‘b’]] is valid
  - len(lst) == 3
    - [2, ‘b’] is one element in list lst

- How to index?
  - [...] all the way down
  - lst[2][1] returns ‘b’

Mutating Lists

- lt = [‘Hello’, ‘world’]
  - How to change lt to: [‘Hello’, ‘Ashley’]
    - Two ways: 1. Build new list or 2. modify list
      1. Concatenation: lt = [lt[0]] + [‘Ashley’]
      2. Index: lt[1] = ‘Ashley’
  - How to change ‘b’ in lt = [1, ‘a’, [2, ‘b’]]?
    - lt[2][1] = ‘c’
Mutating Lists code

```python
lst1 = ['Hello', 'world']
print(lst1)

lst2 = [lst1[0]] + ['Ashley']
print(lst2)
print(lst1)

lst1[1] = 'Ashley'
print(lst1)

lst3 = [1, 'a', [2, 'b']]
print(lst3)

lst3[2][1] = 'c'
print(lst3)
```

OUTPUT:
```
['Hello', 'world']
['Hello', 'Ashley']
['Hello', 'world']
```

Mutating Lists code

```python
lst1 = ['Hello', 'world']
print(lst1)

lst2 = [lst1[0]] + ['Ashley']
print(lst2)
print(lst1)

lst1[1] = 'Ashley'
print(lst1)

lst3 = [1, 'a', [2, 'b']]
print(lst3)

lst3[2][1] = 'c'
print(lst3)
```

OUTPUT:
```
['Hello', 'world']
['Hello', 'Ashley']
['Hello', 'world']
```
Mutating Lists code

```
lst1 = ['Hello', 'world']
print(lst1)
lst2 = [lst1[0]] + ['Ashley']
print(lst2)
print(lst1)
lst1[1] = 'Ashley'
print(lst1)
lst3 = [1, 'a', [2, 'b']]
print(lst3)
lst3[2][1] = 'c'
print(lst3)
```

OUTPUT:
```
['Hello', 'world']
['Hello', 'Ashley']
['Hello', 'world']
['Hello', 'Ashley']
```

Mutating Lists code

```
lst1 = ['Hello', 'world']
print(lst1)
lst2 = [lst1[0]] + ['Ashley']
print(lst2)
print(lst1)
lst1[1] = 'Ashley'
print(lst1)
lst3 = [1, 'a', [2, 'b']]
print(lst3)
lst3[2][1] = 'c'
print(lst3)
```

OUTPUT:
```
[1, 'a', [2, 'b']] 
[1, 'a', [2, 'c']] 
```

WOTO-3 List Mutation