CSc 120
Introduction to Computer Programming
II

CODE EXAMPLES 01
Example 1 😞
def grid_is_square(arglist):
    length = len(arglist)
    i = 0
    for i in range(length):
        x = 0
        if arglist[i][x] == arglist[x][i]:
            return True
        elif arglist[i][x] != arglist[x][i]:
            return False
def grid_is_square(arglist):
    length = len(arglist)
    i = 0
    for i in range(length):
        x = 0
        if arglist[i][x] == arglist[x][i]:
            return True
        elif arglist[i][x] != arglist[x][i]:
            return False
def grid_is_square(arglist):
    length = len(arglist)
    i = 0
    for i in range(length):
        x = 0
        return arglist[i][x] == arglist[x][i]

Simplifying out the Boolean expression makes the bug in the code easier to see
def grid_is_square(arglist):
    length = len(arglist)
    i = 0
    for i in range(length):
        x = 0
        return arglist[i][x] == arglist[x][i]

this initialization has no effect
def grid_is_square(arglist):
    length = len(arglist)
    i = 0
    for i in range(length):
        x = 0
        return arglist[i][x] == arglist[x][i]

we don't need this
def grid_is_square(arglist):
    length = len(arglist)
    for i in range(length):
        return arglist[i][0] == arglist[0][i]

returns on the first iteration
Some code from assg 2 (Pokemon)
def best(list_of_values):
    max_avg = 0.0
    best_items = []

    for value in list_of_values:
        value_avg = compute_average(value)
        if value_avg > max_avg:
            # found a higher best score: re-initialize
            max_avg = value_avg
            best_items = [value]
        elif value_avg == max_avg:
            # found another item with highest score: add to the list
            best_items.append(value)
        else:
            # value_avg < max_avg: ignore and move on
            continue

    return best_items

Q: IS THIS GOOD STYLE?
continue: pros and cons

def foo(...):
  for ...
    if ... :
      ...
    elif ... :
      ...
    else:
      # do nothing
      continue
  return

😊 skips to the next iteration (as intended)

😊 not exactly what a continue is intended for
  – “skip the rest of the loop body”

😊 what if, later, we introduced more code into the loop body?
  – then continue would be a bug

CONSENSUS: continue is not good style here
omit the `else` branch: pros and cons

def foo(...):
    for ...
        if ... :
            ...
        elif ... :
            ...
    return

😊 most direct: doesn't have any "do-nothing" code

😢 some people find the `elif` without an `else` visually weird
   - related issue: someone reading the code might wonder whether the omission of the `else` is a feature or a bug

If you use this style, make sure you add a comment saying that the `else` is deliberately omitted (+ why)
pass: pros and cons

def foo(...):
    for ...
        if ... :
            ...
    elif ... :
        ...
    else:
        # do nothing
        pass
    return

😊 does nothing (as intended)
😊 does not interfere with the loop

CONSENSUS: pass is good style here
avg_dict = {}
for line in mydict.keys():
    for n in range(7):
        sum = 0
        count = 0
        for line2 in mydict[line].keys():
            temp = mydict[line][line2]
            sum += int(temp[n])
            count += 1

Q: IS THIS GOOD STYLE?
avg_dict = {}
for line in mydict.keys():
    for n in range(7):
        sum = 0
        count = 0
        for line2 in mydict[line].keys():
            temp = mydict[line][line2]
            sum += int(temp[n])
            count += 1

Q: Are these names descriptive?
avg_dict = {}
for line in mydict.keys():
    for n in range(7):
        sum = 0
        count = 0
        for line2 in mydict[line].keys():
            temp = mydict[line][line2]
            sum += int(temp[n])
            count += 1

Q: Is this constant throughout the program?
{ 'Fire' :   { 'Charmander': [309, 39, 52, 43, 60, 50, 100],
            'Charmeleon': [405, 58, 64, 69, 65, 90, 80]},
    'Water': { 'Squirtle':   [324, 44, 58, 65, 43, 78, 21],
              'Wartortle': [405, 59, 53, 80, 65, 58, 40] } }

We know what the keys and values represent.
We know what the keys and values represent.

```plaintext
{ 'Fire': { 'Charmander': [309, 39, 52, 43, 60, 50, 100],                                
  'Charmeleon': [405, 58, 64, 69, 65, 90, 80] }, 
'Water': { 'Squirtle': [324, 44, 58, 65, 43, 78, 21],                                   
  'Wartortle': [405, 59, 53, 80, 65, 58, 40] } }
```
Use meaningful names.

```json
{ 'Fire': { 'Charmander': [309, 39, 52, 43, 60, 50, 100], 'Charmeleon': [405, 58, 64, 69, 65, 90, 80]}, 'Water': { 'Squirtle': [324, 44, 58, 65, 43, 78, 21], 'Wartortle': [405, 59, 53, 80, 65, 58, 40] } }
```
The number of pokemon stats (Total, HP, etc.) used does not change.

{ 'Fire': { 'Charmander': [309, 39, 52, 43, 60, 50, 100], 'Charmeleon': [405, 58, 64, 69, 65, 90, 80] }, 'Water': { 'Squirtle': [324, 44, 58, 65, 43, 78, 21], 'Wartortle': [405, 59, 53, 80, 65, 58, 40] } }
Use constants for simple values that do not change.

```json
{ 'Fire': { 'Charmander': [309, 39, 52, 43, 60, 50, 100], 'Charmeleon': [405, 58, 64, 69, 65, 90, 80] }, 'Water': { 'Squirtle': [324, 44, 58, 65, 43, 78, 21], 'Wartortle': [405, 59, 53, 80, 65, 58, 40] } }
```
avg_dict = {}
for line in mydict.keys():
    for n in range(7):
        sum = 0
        count = 0
        for line2 in mydict[line].keys():
            temp = mydict[line][line2]
            sum += int(temp[n])
            count += 1

Exercise 1: Rewrite using descriptive names and a constant.
```python
avg_dict = {}
for poke_type in mydict.keys():
    for n in range(MAX_PSTATS):
        sum = 0
        count = 0
        for poke_name in mydict[poke_type].keys():
            poke_stats = mydict[poke_type][poke_name]
            sum += int(poke_stats[n])
            count += 1
```

Better: *descriptive names and a constant*
Summary

• Avoid this:
  
  if expr:
    return True
  
  else:
    return False

• Use continue (and break) carefully
  – avoid introducing complexity

• When operating on complex data structures:
  – use meaningful names for intermediate results to make the code easier to understand
  – use constants for clarity