# OMIS 30: Intro to Programming (with Python)

## Week 4, Class 2

Introduction to Programming
Instructor: Denis Vrdoljak

Learn programming with Python

# Office Hours

| Instructor | Days available |
|---|---|
| Yuan Wang (our TA) | M 2:30-3:30p, W 9-10a |
| Mike Davis (other section's instructor) | Tu 9:30-10:20a, Th 12-1p |
| Denis Vrdoljak | Tu 3:40-4:40p, W 5-6p |

# Course Topics

1. ~~Computer setup - intros~~
2. ~~Shell - cd, mkdir, move, rename, copy, pwd, touch, echo, nano, vim~~
3. ~~Grep, bash, scripts~~
4. ~~Python Basics - print, input, math, PEMDAS~~
5. ~~Pseudo-code~~, algorithm design, ~~comments~~
6. ~~Loops~~ & Nested Loops
7. Moving files around, input, export
8. Dates, times, epoch, time-series
9. Arrays, ~~lists~~, **dicts**, ~~sets~~, etc.
10. ~~And, or, If, elif~~, try, except
11. **Functions**
12. ~~Strings~~, ~~upper, lower~~, regex
13. Computation time, flops, sorting
14. JSON, dicts, iteritems
15. Pandas
16. Jupyter, virtual environments
17. Web-apps/web-pages
18. Web-scraping
19. Plotting, graphing
20. Git



# Goals from last week

- Understand and be able to use While and For loops
- Know how to write a standalone Python script, and how to run it from the Command Line
- Have made progress on Project 1!



# By the end of this week you should:

- Understand the structure and characteristics of the major container types:
  - Dictionary
  - List
- Understand how to write a function and the advantages of their use
- Be able to say what the main tenets of functional programming are




# Dictionaries

# Dictionaries

- Dictionaries are key:value pairs.
- Think of them like lists, but instead of numeric indexes (0,1,2,3,4…), there are arbitrary strings of text.
- There are multiple syntaxes to make dictionaries
- Keys are unique and immutable (strings)*
- The dictionary is mutable (key:value pairs can be added or removed)

* Dict keys do not have to be strings, though the full definition of what can be used as a key is a bit beyond the scope of this class. Check out this page for more info: https://wiki.python.org/moin/DictionaryKeys

# Dictionary instantiation

We can instantiate dictionaries with a few different syntaxes

```
barn_animalweights = {'Cat':10, 'Dog':25, 'Elephant':2000, 'Giraffe':1000}
```

# Dictionary instantiation

We can lookup values with bracket notation

```
barn_animalweights = {'Cat':10, 'Dog':25, 'Elephant':2000, 'Giraffe':1000}

print(barn_animalweights['Cat'])
```

# Dictionary instantiation

We can lookup values with bracket notation

```
barn_animalweights = {'Cat':10, 'Dog':25, 'Elephant':2000, 'Giraffe':1000}

print(barn_animalweights['Cat'])
```

```
...: print(barn_animalweights['Cat'])
10
```

# Dictionary instantiation

We can use bracket notation to modify, a dictionary

```
barn_animalweights = {'Cat':10, 'Dog':25, 'Elephant':2000, 'Giraffe':1000}
```

```
[10]: barn_animalweights['Dog'] = 45
 ...: print(barn_animalweights)
Cat': 10, 'Dog': 45, 'Elephant': 2000, 'Giraffe': 1000}
```

# Dictionary instantiation

We can also use bracket notation to create a dictionary

```
barn_animalweights = {}
print(barn_animalweights)
#prints an empty dictionary
barn_animalweights['Cat']=10
barn_animalweights['Dog']=25
barn_animalweights['Elephant']=2000
barn_animalweights['Giraffe']=1000
print(barn_animalweights)
```

```
{}
{'Cat': 10, 'Dog': 25, 'Elephant': 2000, 'Giraffe': 1000}
```

# Dictionary exploration

- We made 3 dictionaries
- Consider the bound method ".update()" that can merge 2 dictionaries

```
1 print(farm_dict)
2 print(alt_farm_dict)
3 print(alt_farm_dict2)
```

```
{'donkey': 5, 'horse': 2, 'pig': 10}
{'hippo': 2, 'chicken': 200}
{'horse': 2, 'chicken': 47}
```

# Dictionary mutation

- We change the dictionary by merging a second dictionary with it

```
# use the bound method update to change these in place
farm_dict.update(alt_farm_dict)
```

```
farm_dict
```

```
{'chicken': 200, 'donkey': 5, 'hippo': 2, 'horse': 2, 'pig': 10}
```

# Dictionary exploration

- Get keys, values, or both back

```
4 print(farm_dict.keys())
5 print(farm_dict.values())
6 print(farm_dict.items())
7
```

```
dict_keys(['donkey', 'horse', 'pig', 'hippo', 'chicken'])
dict_values([5, 2, 10, 2, 200])
dict_items([('donkey', 5), ('horse', 2), ('pig', 10), ('hippo', 2), ('chicken', 200)])
```

# Dictionary exploration

- Recall an item based on its keys this can be done with dictionary notation or using the “get” method

```
:  1 print("this uses the get method")
   2
   3 print(farm_dict.get('donkey'))
   4
   5 print("this is dictionary notation:")
   6
   7 print(farm_dict['donkey'])
   8
```

```
this uses the get method
5
this is dictionary notation:
5
```

# zip()

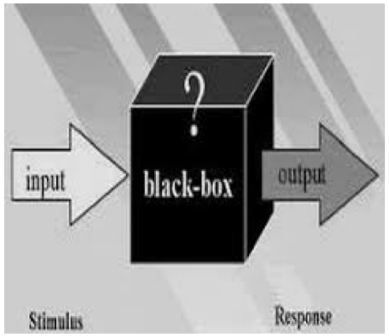
- Used to pair up the elements of two lists (or other iterable) based on shared index
  - odd = (1,3,5), even = (2,4,6)
  - >> print(list(zip(odd, even)))

    [(1,2),(3,4),(5,6)]
- Can also be used with dictionaries:
  - students = [“Matt”, “Jane”, “Bob”], grades = [82, 97, 70]
  - >> print(dict(zip(names, grades)))

    {“Matt”:82, “Jane”:97, “Bob”:70}

# What is a function?

- Think of a function as a machine that does a specific task
- The function ‘machine’ has a defined  input and and output
- You put something known in and get something known out



# What is a function?

**function ()**

KNOWN INPUT -> PROCESS -> EXPECTED OUTPUT

- If you put the wrong thing in what will happen?



# Definition

**A function is an object that:**

1. Takes in pre-specified data as arguments
2. Processes the data in pre-determined way
3. Returns the data after processing

This way of programming minimizes "**unintended side effects**"

# Function syntax

Terminate the function def with a ":"

Declare the function definition

```
def skill_to_expert(argument):
    """Takes the name of a skill and tells you what
    to call the expert at that skill

    USE EXAMPE : out=skill_to_expert('art')
    ARGUMENTS: string with skill name"""

    expert = argument +  'ist'

    return expert
```

The Docstring gives information to the user and can be called with help ()

**use """ triple quotes**

The input is processed in the body of the function

**NOTE the indent of 4 spaces or one tab to delimit the body of the function**

Return specifies the output

# Make the function object

- Before we execute the function it is just an object (like a book or a cell phone)
- It is just sitting in object space doing nothing!
- When we print it Python tells us its type (function) and memory address

```
4  def skill_to_expert(argument):
5      """Takes the name of a skill and outputs expert name
6      USE EXAMPE : out=skill_to_expert('art')
7      ARGUMENTS: string with skill name"""
8
9      expert = argument +  'ist'
10     return expert
11
```

```
1  print(skill_to_expert)
```

```
<function skill_to_expert at 0x10be41ea0>
```

# Start with a working example

Since there is a good example in the documentation, we can start with that

Make sure everything is in working as expected

```
4 out=skill_to_expert('art')
5 print (out)
```

```
artist
```

# Moving forward

1) We verified that the example works
2) Try it with a different word
3) Remove the variable assignment so it simply outputs to the workspace

```
7 skill_to_expert('BBQ')
8
```

```
'BBQist'
```

** NOTE the output is not bound to any variable so we can see it but then it's GONE!

This is a good way to test rapidly your function but if you want to use it again bind it to a variable like we did with "out" (on the last slide)

# Functions have expected inputs

- If you send the wrong variable type into a function, Python emits an error and tells you specifically what went wrong (read it from the bottom up)
  - We'll get to errors more later!

```
  4 skill_to_expert(1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-2399ce3bc3fc> in <module>()
      2
      3
----> 4 skill_to_expert(1)

<ipython-input-6-ad7ada7d86cd> in skill_to_expert(argument)
      9     ARGUMENTS: string with skill name"""
     10
---> 11     expert = argument +  'ist'
     12
     13     return expert

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```



# What is Functional Programming

**Functional Programming is a style of programming**

- The main tenet of Functional Programming is that the programmer has complete control over what occurs in the program.

- The elimination of unintended **side effects**

- **Purity** of code
    - a pure function takes a **defined input** and returns a **defined output**



# What is Functional Programming

**As show in the example above, the function clearly does one thing:**

skill_to_expert('skill') -> 'skillist'

skill_to_expert(1) -> TypeError!

- In other words, it avoids unintended side effects and does a defined job



# Benefits of functional programming (discuss)

**Modularity** - it should be easy to remove and replace functions without affecting other code areas

**Reusability -** clear definition makes reuse in other context possible

**Abstraction -** the details of the function internals are obscured allowing the are programmer to think about higher order code processes

**Scalability** - clear definition of task blocks allows replication and scaling

**Ease of  troubleshooting -** broken code can be traced to single isolated functions

# Learn about the function - Docstrings matter!

- We can call the help() function:
  - Takes your function name as an argument and returns its docstring
- Hopefully there is a good doc string with a working example to get us started!

```
1 help(skill_to_expert)
```

```
Help on function skill_to_expert in module __main__:

skill_to_expert(argument)
    Takes the name of a skill and tells you what
    to call the expert at that skill

    USE EXAMPE : out=skill_to_expert('art')
    ARGUMENTS: string with skill name
```

# More Functions

```
1  import random
2  # random generates pseudo-random numbers from a given set
3
4  def coin_flip(tosses):
5      '''coin_flip takes an integer number of tosses
6      and outputs a list of random outcomes'''
7
8      outcome = []
9      coin = ['heads','tails']
10     for toss in range(tosses):
11         outcome.append(random.choice(coin))
12     return(outcome)
13
14 coin_flip(5)
```

What's the expected output?

# More Functions

```
1  import random
2  # random generates pseudo-random numbers from a given set
3
4  def coin_flip(tosses):
5      '''coin_flip takes an integer number of tosses
6      and outputs a list of random outcomes'''
7
8      outcome = []
9      coin = ['heads','tails']
10     for toss in range(tosses):
11         outcome.append(random.choice(coin))
12     return(outcome)
13
14 coin_flip(5)
15
```

```
['tails', 'tails', 'heads', 'tails', 'heads']
```

# Homework 4.2

- Redo the HW from week 2 as a function.
- We reviewed this last week, so check on GitHub if you don't remember what the solution should look like.
- def simple_csv_parser(data_as_string):

  """ This function should take in a string. The string
  will be csv data (rows separated by newline characters,
  values separated by commas within the rows).
  The output should be a list of lists of strings. A list, of lists (each list representing 1 row) of strings (each string being 1 value
  In the csv)."""

  Details and extra credit on GitHub

# Appendix

- Reserved keywords in Python:
  - https://pentangle.net/python/handbook/node52.html
  - DO NOT USE THESE as VARIABLE NAMES!
- Built-in Functions:
  - https://docs.python.org/3/library/functions.html
  - Review/Reference Material:
  - https://developers.google.com/edu/python/lists

# Appendix

- Reserved keywords in Python:
  - https://pentangle.net/python/handbook/node52.html
  - DO NOT USE THESE as VARIABLE NAMES!
- Built-in Functions:
  - https://docs.python.org/3/library/functions.html
  - Review/Reference Material:
  - https://developers.google.com/edu/python/lists
- Dos/Windows vs Unix/Linux:
  - https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Step_by_Step_Guide/ap-doslinux.html
  - You should be familiar with the basic commands for navigating around the file structure and modifying/creating files/folders. You should be aware of the harder to remember ones (like grep and vi) so you know what to Google when you need them!