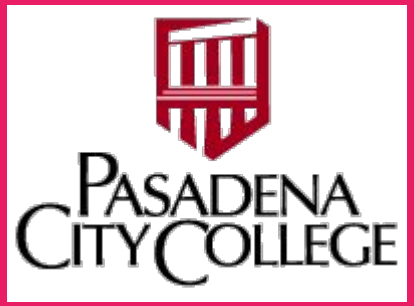



# CIS 112

## Intro to Programming Using Python

Module 1 Part 1

# Agenda for the Day!

---

- Topics
- Syllabus
- Projects
  - Review of CIS 012

# Topics Covered in this Course

# About CIS 112:

---

- Course description:
  - Application development using advanced Python programming techniques. Topics include the **advanced Object-Oriented programming**, advanced **functions** and **modules**, **packages**, **process management**, **network programming**, **web programming**, **GUI** (Graphical User Interface) programming, and testing principles and techniques.
  - Preparation for the **Python Institute Certified Associate in Python Programming (PCAP)** and **Python Institute Certified Professional in Python Programming (PCPP)** exams

# Now about those exams...


- **PCPP1™ - Certified Professional Python Programmer Level 1 certification** is the first of the two-series General-Purpose Programming track professional credentials from the OpenEDG Python Institute
- The PCPP1™ certification shows that the individual demonstrates proficiency in the advanced use of classes and OOP features present at the hear of Python programming; knows, understands, and implements the coding conventions, design practices, and standards for code writing; knows how to build a GUI application using the most essential tools and toolkits, conventions, and elements of event-driven programming; understands the basic concepts of network programming and what data formats are used in client-server communication, knows how to use sockets and HTTP methods, and is able to create a simple REST client; knows how to use some of the most important Python Standard Library modules for file processing and interacting with a program's environment.

# Topics: Review and OOP

---

- Part 1: Review of Introduction to Python (Weeks 1 and 2)
  - Environment
  - Datatypes
  - Control Flow
  - Functions
- Part 2: Advanced use of classes and modelling real-life problems in the OOP categories (Weeks 3 and 4)
  - Classes overview,
    - instances,
    - attributes,
    - methods;
    - class and instance data;
    - shallow and deep operations;
    - inheritance and polymorphism;
    - extended function argument syntax and decorators;
    - static and class methods;
    - attribute encapsulation;
    - composition and inheritance;
    - advanced exceptions;
    - copying object data;
    - serialization;
    - metaclasses,
  - Modules

# More Topics: Standardization and GUI

---

- Part 3: Best practices and standardization (Weeks 5 and 6)
  - PEP8,
  - PEP 257,
  - code layout,
  - comments and docstrings,
  - naming conventions,
  - string quotes and whitespaces,
  - programming recommendations
- Part 4: GUI programming (Weeks 7 and 8)
  - events,
  - widgets,
  - geometry,
  - tools and toolkits,
  - conventions,

# More Topics: Networking, File Processing, and Communication

---

- Part 5: Network programming
  - network sockets,
  - client-server communication,
  - JSON and XML files in network communication,
  - HTTP methods,
  - CRUD,
  - building a simple REST client,
- Part 6: File processing and communicating with a program's environment
  - processing files:
    - sqlite3,
    - xml,
    - csv,
    - logging,
    - configparser;
- Part 7: Communication:
  - os,
  - datetime,
  - io,
  - time

# Syllabus

# Syllabus

---

- Video Summaries: 10%
- Programming Labs: 40%
- Midterm Programming: 25%
- Final Programming: 25%

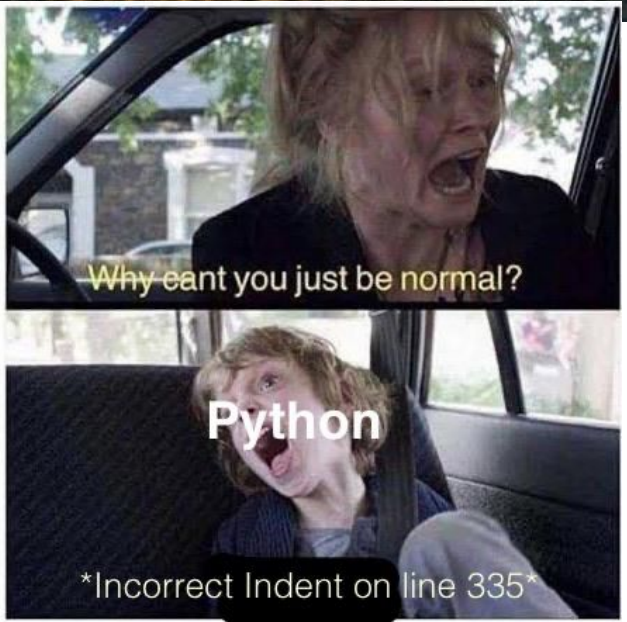
# Review of CIS 012

# Algorithms and Program Design

- Programming is the process of abstracting, generalizing, and systematic solution-design
  - Variables allow for abstraction (solve the problem form, not the specific instance
  - Generally almost every problem is reducible to the creation, maintenance and management of data
  - Algorithmize
    - Break a big problem into a series of discrete steps, each of which we can encode
  - Then we introduce control flow: sometimes we want more complex program flows
    - Branching: we use if/then statements to direct program flow based on external factors
    - Loops: we can control the number of times a sequence is run

# Formatting and Whitespace

- Whitespace is meaningful in Python especially indentation and placement of newlines
- Use a newline to end a line of code
- Use \ when must go to next line prematurely
- No braces {} to mark blocks of code, use consistent indentation instead
    - First line with less indentation is outside of the block
    - First line with more indentation starts a nested block
- Colons start of a new block in many constructs, e.g. function definitions, then clauses
- The basic printing command is print()

# Assignment

---

- Binding a variable in Python means setting a name to hold a reference to some object
  - Assignment creates references, not copies
- Names in Python do not have an intrinsic type, objects have types
  - Python determines the type of the reference automatically based on what data is assigned to it
- You create a name the first time it appears on the left side of an assignment expression:

  ```
  x = 3
  ```
- A reference is deleted via garbage collection after any names bound to it have passed out of scope
- Assignment is = and comparison is ==

# Variables

- So why bother?
  - One = 1
  - Two = 2
  - Three = One + Two
- Variables as references don’t simply refer to individual values, but can refer to complex data elements (e.g., lists) or even entire datasets, or functions, or libraries
- Oftentimes we need to make continuous iterative edits to a variable without explicitly knowing start and end values. This tells the program what element to act on regardless of the underlying value

# DataTypes and Inputs

# Datatypes

---

- Python supports both basic and complex datatypes
- Each type is it’s own class: it stores a particular format of information, and has certain allowable operations
  - Python is dynamically typed, which means types are assigned on the fly, and modified
- Python also includes complex types for collections of data. These are based on mutability (the ability to edit without overwriting), and indexability (if and how elements can be referenced

# Basic Datatypes

- Integers (default for numbers)
  - `z = 5 / 2  # Answer 2.5`
  - `w = 5 // 2  # Answer 2, integer division`
- Floats
  - `x = 3.456`
- Strings
  - Can use “” or ‘’ to specify with “al
    == ‘abc’
  - Unmatched can occur within the string: “matt’s”
  - Use triple double-quotes for multi-line strings or strings than contain both ‘ and “ inside of them: “““a‘b“c”””

**Sarah J. Hass**
@tacko_belle

my eye doctor told me to look ahead for two seconds and then said "perfect!" and that is the exact ratio of task to praise I need

# The Boolean Datatype

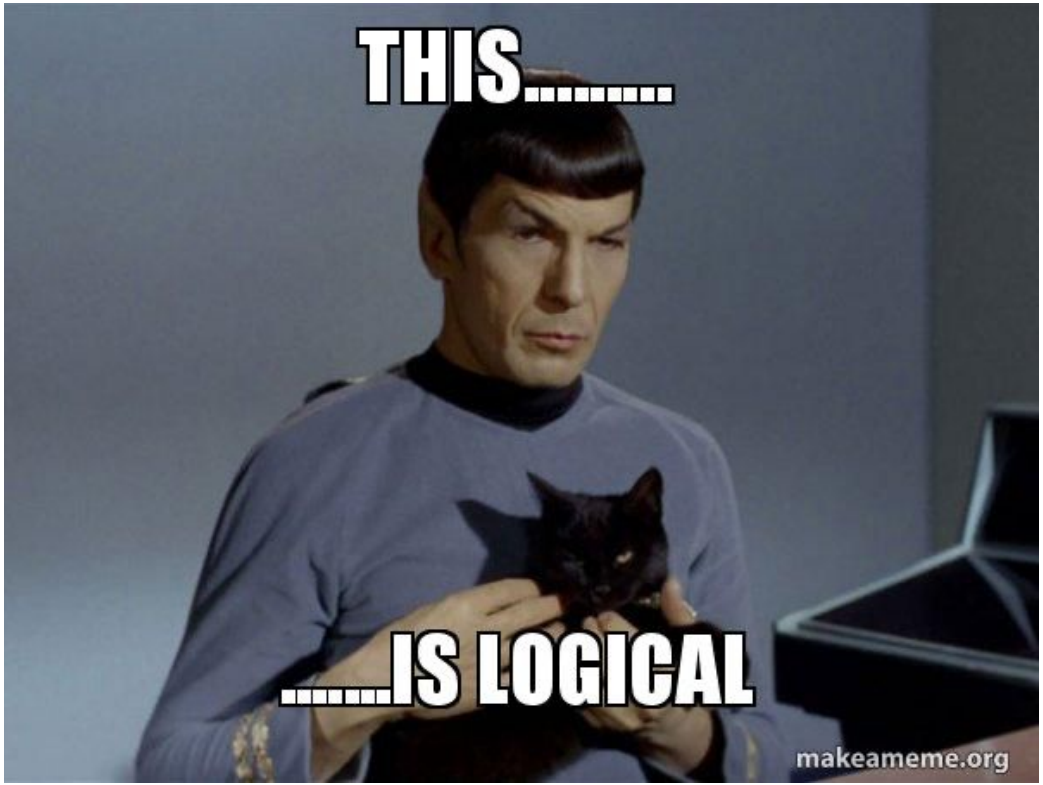

- In addition to merely evaluating the result, we can store that result as a unique datatype: Boolean
- Booleans like other variables can be assigned directly or as the result of a calculation

```
[1]  result = True
     otherResult = False
```

# Logic!

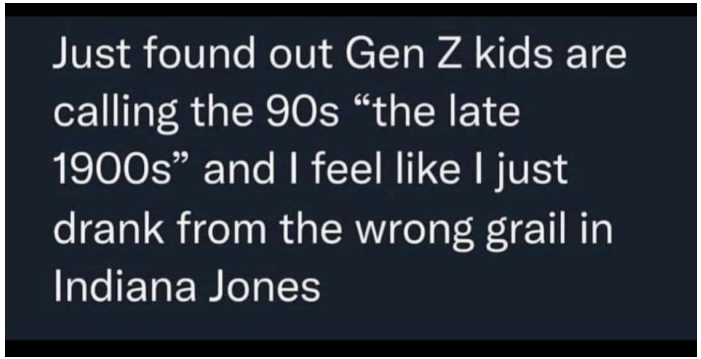
- In addition to mathematical operations, Python can assess the ‘truth’ of a mathematical expression
  - ‘Is this expression mathematically accurate’
- Logical operators are both key words (and, or, not) and  symbols

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`



# Complex Data Storage

# Lists!




- The workhorse of Python data structures!
  - Basically the default due to its functionality
- To create simply encase your desired data in square brackets, with data elements comma-separated
  - Numbers = [1,2,3,4]
- Characteristics:
  - Indexable
  - Mutable
  - Flexibly-typed
- Can perform list-level operations

# Tuples!


- Just like lists, but… **Immutable**!
- Declared similarly, but this time with parentheses
  - Numbers = (1,2,3,4)
- Advantages in speed and permanent references

# Sets!


- Maybe the least useful data collection type
- Unordered (so cannot be referenced!), multi-typed, and immutable
- Can be made through curly brackets:
  - Numbers = {1,2,3,}
- Only truly unique aspect of sets are all values have to be unique (so can’t be replicated)
  - Can be useful as a reference (is this value IN this set)

# Dictionaries!

- A ***dictionary*** is a Python collection used to describe associations or relationships between elements
  - Its values are mutable, and ordered!
- A dictionary is ordered, but not indexed, meaning that it associates (or "maps") keys with values.
  - A ***key*** is a term that can be located in a dictionary, such as the word "cat" in the English dictionary.
  - A ***value*** describes some data associated with a key, such as a definition. A key can be any immutable type, such as a number, string, or tuple; a value can be any type.
- A dict object is created using ***curly brackets*** { } to surround the ***key:value pairs*** that comprise the dictionary contents.
  - Ex: Numbers = {'one': 10, 'two': 2, ‘three’: 3}
- Can reference using the key, and update the value

# Control Flow

# Branching! The fate of our Program at the Tips of our Fingers


- We've already introduced the concept of programs as universal solutions (where we use placeholder variables to allow users to plug in specific values for solutions) but we're going to extend that logic further…
  - Previously we would always do a consistent set of things (e.g., adding one number to another) but would simply change the inputs depending on the specifics of the user
- Now we introduce the idea that we can also control what steps a program takes, depending on the particular values fed into it
  - E.g., does your HVAC system turn on heat, AC, or stay off? Well that depends on the temperature it reads…
- Logic statements enable this functionality!
  - We can ask our program to evaluate whether something is true or not and then act based on the result!

# Introducing Conditionals!

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

- The ‘if’ keyword triggers a condition-check immediately after-which we put in the condition to evaluate
- IF (and only if) it returns true, we then execute the code immediately below it
- ‘elif’ allows us to sequentially evaluate a set of conditions
- ‘else’ is our fallback. If nothing has proven true we return a default result

# `while` LOOPS

```
while <condition>:
    <expression>
    <expression>
    ...
```

- `<condition>` evaluates to a Boolean
- if `<condition>` is `True`, do all the steps inside the while code block
- check `<condition>` again
- repeat until `<condition>` is `False`

# for LOOPS

```
for <variable> in range(<some_num>):
    <expression>
    <expression>
    ...
```

- each time through the loop, <variable> takes a value
- first time, <variable> starts at the smallest value
- next time, <variable> gets the prev value + 1
- etc.

# `for` VS `while`

## `for` VS `while` LOOPS

`for` loops

- **know** number of iterations
- can **end early** via `break`
- uses a **counter**
- **can rewrite** a `for` loop using a `while` loop

`while` loops

- **unbounded** number of iterations
- can **end early** via `break`
- can use a **counter but must initialize** before loop and increment it inside loop
- **may not be able to rewrite** a `while` loop using a `for` loop

# `break` STATEMENT

- immediately exits whatever loop it is in
- skips remaining expressions in code block
- exits only innermost loop!

```
while <condition_1>:
    while <condition_2>:
        <expression_a>
        break
        <expression_b>
    <expression_c>
```

- A way of thinking about a break statement is ‘we’ve found what we want, no need to do anything further

# continue STATEMENT

---

- Continue statements will end the current branch of a loop advancing it to the next option in the loop **Without exiting**
- You can think of it as saying:
  - ‘The solution isn’t here. No need to evaluate this option further, continue on to the next one!’

# Nested Loops

- Sometimes we will want to run multiple nested loops to evaluate sub-options or choices within them by running additional loops

- Example 1: I want to scour a listing of restaurants in my neighborhood to find one if they have ‘Sweet potato Fries’ on their menu
- For each restaurant in my list: (high-level loop where i loop through every restaurant)
    - for each item on their menu (Nested loop where I take each restaurant’s menu and loop their items looking for a match)
        - if item == ‘Sweet potato Fries’:
            - Print restaurant
            - Break (Once I find a match, print it and break!)

- Example 2: I want to figure out how many unique combinations of coins i can use to make a distinct sum (e.g., how many unique ways can I make $1 out of quarters, dimes, and nickels
- For each quarter (0 to target sum)
    - For each dime (0 to target sum)
        - For each nickel (0 to target sum)