Functional Programming (1930s - present):

* all about evaluating mathematical formulas using function calls
* no state changes or side effects
* Examples: lambda calculus, logic programming, Lisp, Scheme, Haskell

Imperative Programming (50s - 70s):

* focuses on global state changes
* a variable contains a value, an assignment statement changes it
* a print statement sends a value to output
* a "go to" transfers control to another statement
* "verbs" are the most important thing
* Functions / subprograms not really emphasized - used for reuse
* Examples: FORTRAN, COBOL, Algol, Basic

Procedural Programming:

* Similar to Imperative, but state changes are localized within subprograms
* State changes are communicated to other subprograms by parameters (arguments, return values)
* break down a task into variables, data structures, and subprograms
* you use subprograms to operate on data structures

Structured Programming (70s-80s):

* Focus on making programs easier to write, debug, and understand by use of subprograms, block structures, and for/while loops
* "go to" statements are blasphemous
* Ex: Pascal, C

Object-Oriented Programming (80s - present):

* Imperative in style, but features added to support Objects
* Ex: Smalltalk, Simula, C++, Python, Visual Basic, Java, Ruby
* define *objects* that expose behavior (*methods*) and data (*attributes*) using well-defined interfaces
* bundle everything together, so that an object only operates on its own attributes using methods

***Encapsulation:*** hiding implementation details of a class from other objects, so other objects cannot use details from another object

***Abstraction:*** simplifying complex reality by modeling classes appropriate to the problem, so the output does not need to show how it got there

***Inheritance:*** a way to define new classes using parts of classes that have already been defined.

***Polymorphism:*** the process of interpreting an operator or function in different ways for different data types.

Dunder: str(self),add(self,other),sub(self,other),mul(self,other),trudiv(self,other)/,floordiv(self,other)//,mod(self,other)%,

eq(self,other)==

list methods: .append(item), .insert(i,item), .pop(), .pop(i), .sort(), .reverse(), del alist[i], .index(item), .count(item), .remove(item)

string methods: .count(item), .center(w), .ljust(w), .rjust(w), .lower(), .find(item)- returns index of first occurrence, .split(schar)

isinstance(item, object)- tests if item is an instance of object

Order of magnitude: O(1), O(log n), O(n), O(n log n), O(n^2), O(2^n), O(n!)

O(n log n): sorting

O(1): indexing, index assignment, append, pop()

O(n): pop(i), insert(i, item), iterate through, contains, del slice, set slice, reverse

O(k): getslice() “a[3:6]”, concatenate “[size n][size k] = [size n+k]” because you append k items

Checking off: O(n^2), Sort and Compare: O(n log n), Brute Force: O(n!), Count and Compare: O(n)

Deep Equality: comparison of the contents, Shallow Equality: comparison of the pointers

Mutable: can be changed after created

* list, dictionary, almost all user defined classes

Immutable: cannot be changed after created

* ints, floats, strings

Turtle Graphics:

* import turtle
* ttl = turtle.Turtle()
* screen = turtle.Screen()
* screen.setup(x,y)
* .speed(speed), .penup(), .pendown(), .goto(x,y), .circle(r), .fillcolor(“color”), .begin\_fill(), .end\_fill(), .forward(d), .left(d), .right(d), .backward(d), .setx, .sety