1. OOP8: Inheritance, Subtype Polymorphism, Dynamic Dispatch (5 min)
   Inheritance is used for a subclass to be able to inherit properties and reuse behaviors of a parent class. Inheritance doesn't auto imply polymorphism or dispatch. A subtype polymorphism example would be having a parent class being abstract and generalizable, so that you can pass a subclass of it into a function as ie. a parameter and be fine with it. Dynamic Dispatch is where you determine which version of a polymorphic method to invoke.

2. OOP9: Subtype Polymorphism, Dynamic Dispatch, Dynamically Typed Languages (5 min)
   Subtype polymorphism relies on static typing, where the compiler verifies that an object conforms to a specific type. This allows the same code to work with objects of different types through a shared interface or base class. In dynamically-typed languages, the type of a variable is determined at runtime, and there is no compile-time enforcement of type constraints. On the other hand, dynamic dispatch can occur in dynamically-typed languages. In fact, dynamic dispatch is a core mechanism of method resolution in such languages because method calls are resolved at runtime based on the object's type and its available methods.

3. OOP11: Liskov Substitution Principle (5 min)
   LSP applies to dynamically-typed languages in theory because it ensures substitutability. However it depends very heavily on runtime behavior, as if you substitute all superclass objects A with subclass object B and A had a certain trait to itself only and then you try accessing the trait in B, you get an error.

4. CNTL1: Short Circuiting (5 min)
   Parenthesis get evaluated first. && has a higher precedence than || so you can effectively put a parenthesis around the two operands that are affected by the &&. For && if the first operand evaluates to false the other one is not calculated. For || if the first operand results true the second one isn't calculated.

5. CNTL2: Iterators, Iterator Classes, Generators, First-class Iteration (31 min)
   a.

```
def generator(self):
        for head in self.array:
            current = head
            while current is not None:
                yield current.value
                current = current.next
```

   b.

```
class htIterator:
def __init__(self, hash_table):
    self.array = hash_table.array
    self.bucket_index = 0
    self.current = None

def __iter__(self):
    return self

def __next__(self):
    # Find the next node in the hash table
    while self.current is None and self.bucket_index <
len(self.array):
        self.current = self.array[self.bucket_index]
        self.bucket_index += 1
    if self.current is None:
        raise StopIteration
    value = self.current.value
    self.current = self.current.next
    return value
```

c.

```
ht.iter = lambda: htIterator(ht)
for item in ht:
    print(item)
```

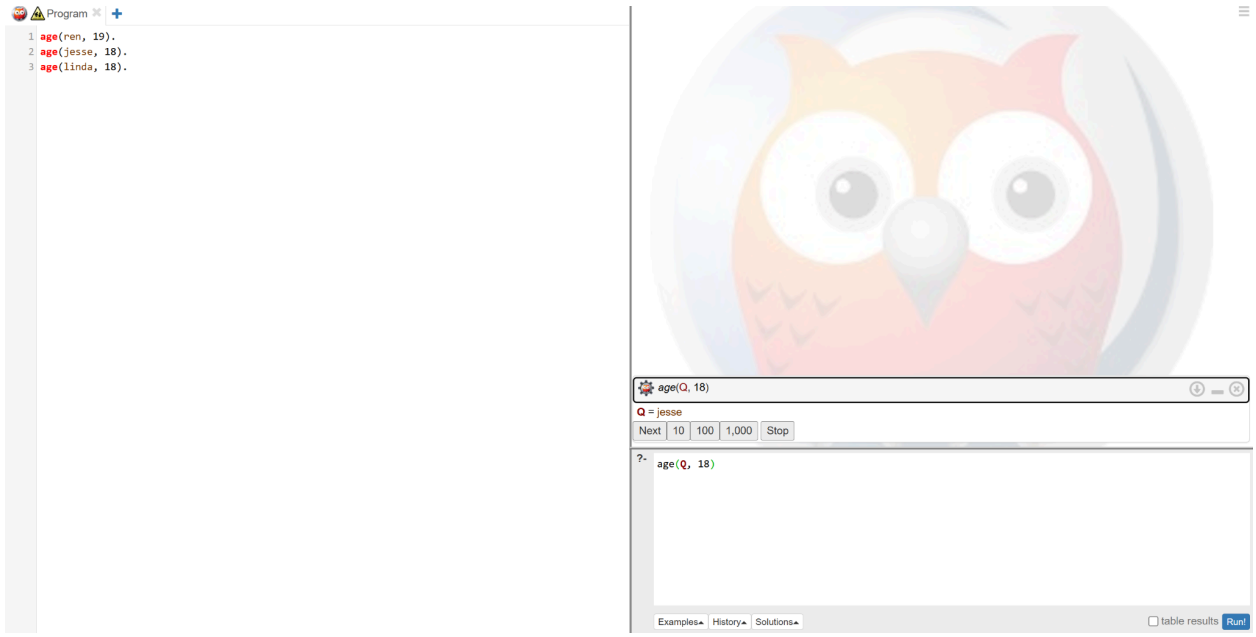
d.

```
iterator = iter(ht)
while True:
    try:
        item = next(iterator)
        print(item)
    except StopIteration:
        break
```

6. CNTL3: Async Programming (5 min)
   Main start
   Foo start
   Bar start
   Sync call

Main middle
Foo end
Bar end
Main end Foo result Bar result

7. PRLG1: Swish Prolog (5 min)



8. PRLG2: Facts (10 min)
    a. X = green
    b. false
    c. Q = tomato
    d. Q = celery, R = green

9. PRLG3: Facts, Rules (10 min)
    a. likes_red(Person, Food) :- likes(P, F), food(F), color(Food, red).
    b. likes_foods_of_colors_that_menachen_likes(P) :- likes(menachen, F), color(F, C), likes(P, OF), food(OF), color(OF, C).

10. PRLG4: Facts, Rules, Recursion, Transitivity (10 min)
    reachable(A, B) :- road_between(A, B)
    reachable(A, B) :- road_between(A, C), C A \= C, reachable(C, B).

11. PRLG5: Unification (5 min)
    <u>UNIFIES</u>:
    foo(bar, bletch) with foo(X, bletch) → X = bar

foo(Z, bletch) with foo(X, bletch) → Z = X
foo(X, bletch) with foo(barf, Y) → X = barf, Y = bletch
foo(bar, bletch) with foo(bar, bletch, barf) → Does not unify
foo(bar, bletch) with foo(barf, Y) → Y = bletch
foo(bar, bletch(barf, bar)) with foo(X, bletch(Y, X)) → X = bar, Y = barf
foo(barf, Y) with foo(barf, bar(a, Z)) → Y = bar(a, Z)
foo(Z, [Z | Tail]) with foo(barf, [bletch, barf]) → Z = barf, Tail = [barf]
foo(Q) with foo([A, B | C]) → Q = [A, B | C]
foo(X, X, X) with foo(a, a, [a]) → X = a

<u>DOESN'T UNIFY</u>:
foo(bar, bletch) with foo(bar, bletch, barf)
foo(Z, bletch) with foo(X, barf)

12. PRLG6: Lists, Recursion (10 min)

```
insert_lex(X, [], [X]).
insert_lex(X, [Y | T], [X, Y | T]) :- X =< Y.
insert_lex(X, [Y | T], [Y | NT]) :- X > Y, insert_lex(X, T, NT).
```

13. PRLG7: Lists, Recursion (10 min)

```
count_elem([], Acc, Acc).
count_elem([Hd | T], Sum, Total) :-
        Sum1 is Sum + 1,
        count_elem(T, Sum1, Total).
```

14. PRLG9: Lists, Recursion (15 min)

```
append_item([], Item, [Item]).

append_item([Hd | T], Item, [Hd | Result]) :-
        append_item(T, Item, Result).
```