Outline for October 6, 2025

Reading: §2, 5, 6.1–6.8 **Due:** Homework 1, due October 15, 2025

1. Decision structures [*if0.py*]
   (a) If statement
   (b) Executes once, based on condition
   (c) Syntax
2. Conditions
   (a) Resolves to boolean value
   (b) Literal booleans: True (1), False (0)
   (c) Testable as `true` or `false`
   (d) Relational operators
      i. Use two arithmetic expressions connected with relational operators to create a boolean
      ii. Relational operators: $>$, $>=$, $<$, $<=$, $==$, $!=$
      iii. Precedence: resolved after arithmetic operators
      iv. `6 > 2 + 3`; `"UCD" == "Sac State"`
3. Two-way decisions [*if1.py*]
   (a) if . . . else statements
   (b) One condition, two possible code blocks
   (c) Syntax
   (d) else very powerful when the positive condition is easy to describe but not the negative
   (e) String comparison example
4. Multi-way decisions [*if2.py*]
   (a) Can execute code based on several conditions
   (b) `elif` (else if)
   (c) Syntax
   (d) *else* only reached if all previous conditions false
   (e) Nested if statements
5. Conditional expressions [*condexp.py*]
6. Iteration
   (a) Definite loops: execute a specific (definite) number of times
   (b) Indefinite loops: execute until a general condition is false
7. For loops
   (a) General form: `for i in` *`iterator`*
   (b) *Iterator* is either list or something that generates a list
   (c) Very common form: `for i in range(1, 10)`
8. While loops [*while.py*]
   (a) Contrast with `for`
9. `range()` in detail [*for.py*]
   (a) `range(10)` gives 0 1 2 3 4 5 6 7 8 9

   (b) `range(3, 10)` gives 3 4 5 6 7 8 9
   (c) `range(2, 10, 3)` gives 2 5 8
   (d) `range(10, 2, -3)` gives 10 7 4
10. `continue` and `break` statements in loops [*loop1.py*]
11. Exception `Keyboard Interrupt` — user hit the interrupt key (usually control-C)
12. Program: counting to 10 [*toten.py*]
13. Program: sum the first 10 squares [*sumsq.py*]
14. Program: Fibonacci numbers [*fib.py*]
15. `import` statement
   (a) `import math` [*hypotnoex.py*]
   (b) Need the "math." before "sqrt"
   (c) `from math import sqrt` [*hypotnoex1.py*]
   (d) Do not need the "math." before "sqrt"
   (e) Now add in exception handling [*hypotex.py*]
16. Full version of the hypotenuse program [*pythag1.py*]
17. Exception `ValueError` — built-in function or operation applied to operator with illegal value
18. Functions [*hello.py*]
   (a) What functions are
   (b) Defining them
   (c) Using them
19. Quick look at using them [*quad.py*]
   (a) Passing values to functions
   (b) Returning values from functions
20. In more detail: passing values to functions [*args.py*]
   (a) Formal parameters in subject definition
   (b) Actual parameters (arguments)
   (c) Matching arguments to formal parameters
   (d) Local variables
21. In more detail: how Python does function calls [*quad.py*]
   (a) Caller suspends execution at point of call, remembers where it left off
   (b) Formal parameters assigned values from actual parameters
   (c) Execute function body
   (d) Return control to where caller left off
22. Refactoring code
   (a) Compute the perimeter of a triangle [*peri0.py*]
   (b) Collapse similar statements: make the distance between 2 points a function [*peri1.py*]
   (c) Collapse similar statements: make the prompts a function [*peri2.py*]
   (d) Refactor for clarity only: make the perimeter computation a function [*peri3.py*]
   (e) Add error checking: "peri0.py" done right [*peri-c.py*]

23. Add error checking: "quad.py" done right [*quad-c.py*]
24. Sequences
    (a) Sequences are a series of values in a particular order
    (b) In Python predominantly strings and lists but also sets and tuples
25. Strings
    (a) Sequence of characters (characters are strings of length 1)
    (b) Strings are immutable; really important for functions
26. Basic string operations
    (a) `+`, concatenation for strings
    (b) `*`, repetition repeats given value
    (c) `len()` returns length of sequence
    (d) `s in str` returns `True` if `s` is a substring of `str`, `False` otherwise
27. Indexing, `var[position]`
    (a) Count from 0 to `len(var)`-1
    (b) Position can be a negative number to count from right
28. Assignment with indexing doesn't work as strings immutable
    `x = 'hEllo'; x[1] = 'e'` produces an error
29. Slicing, `var[start:end]`
    (a) Value at index end not included in slice
    (b) If omitted, starting value defaults to 0 and ending value defaults to last index + 1
    (c) Can use negative index
30. Looping over strings: `for i in str`
31. Example program [*strstuff.py*]