Chapter 6 Functions

Lesson Points

questions are based on these points

- functions are an important tool for building sophisticated programs
- used to reduce code duplication and make the program easier to maintain and understand
- functions are subprograms/ a small program inside of a program
- the part of the program that creates a function is called function definition
- when a function is used in the program we say that the function was called/ invoked
- can be used multiple times within the program
- parameters are variables that are initialized when the function is called
- we need to supply the functions with parameters so that it can be invoked
- scope of variables / refers to the places in a program where a given variable may be referenced
- the variables used inside one function are local to that function even if they have the same name as variables that appear inside of another function
- When python calls a function, it initiates a four step process:
    1. the calling program suspends execution at the point of the call
    2. the formal parameters of the function get assigned the values specified by the actual parameters
       in the call
    3. the body of the function is executed
    4. control returns to the point just after where the function is called
- when a function has several parameters, the actual parameters are matched up with formal parameters by position

- needs a return statement to return a value

- when python encounters a return statement, it exits the function and returns control to the point where the function is called while the value provided in the return statement are sent back to the caller as an expression result
- all functions return a value whether or not it contains a return statement
- functions without a return statement always hand back a special object, denoted as none
- passing parameters by reference / allowing variables themselves to be sent as parameters / does not work with python
- if the value of the variable is a mutable object / like a list / then changes to the state of the object will be visible to the calling program

Chapter 7 Decision Structures

- decision structures / statements that allow a program to execute different sequences of instructions for different cases / allows program to chose appropriate course of action
- if statements / if condition is true, body is executed. if conditions are false, body is skipped
- if statements/ one way/simple decisions
- conditions may compare numbers or strings
- when comparing strings / ordering is lexicographic / strings are put in alphabetical order according to ASCII codes
- conditions are Boolean expressions/ when evaluated, it produces a value of either true or false
- if/else statements / when python encounters this structure, it first evaluates the condition. If it is true, the

statement under if are executed. if the condition is false, the statements under else are executed

- body of if/else statements can include other if/else statements
- putting one compound inside another is called nesting
- if/elif/else / evaluates if statement then goes to evaluate elif statements. if none of conditions are true/ evaluates else statements
- exception handling / programmers can write code that catches and deals with errors that arise when the program is running
- tells user the error in their numbers or program
- a single try statement can have multiple except clauses
- python searches through all exceptions to find one that matches
- exceptions are objects
- when python encounters a try statement, it attempts to execute the statements inside the body
- if these statements execute without error, the control passes to the next statement after try which is except
- if an error occurs inside the body, python looks for an except clause by matching error type
- if a suitable except is found, the handler code is executed
- instead of crashing, the exception handler catches the error and prints a message
- try and except statements can be used to catch any error
- errors include: value error, name error, type error

- if an error occurs, python will try each except block in turn looking for one that matches the type error
- the last except acts like an else ad will be used if none of the others math
- positive side of not following sequential program / can develop more sophisticated software.
- downside / designing sophisticated software is much harder
- compound condition (if x1>= x2 >= x3) allowed in python but not other programs
- one way to avoid redundant tests is through decision trees
- decision tree strengthen efficiency

. Chapter 8 Loop Structures and Booleans

- indefinite or conditional loop / keeps iterating until certain conditions are met/ no guarantee ahead of time regarding how many times the loop will go around
- an indefinite loop is implemented using a while loop
- ```
  while <condition>:
  <body>
  ```
- condition is a Boolean statement
- the body of a while loop executes repeatedly as long as the condition remains true
- when the condition is false, the loop terminates
- while loop needs programmers to initialize variable before the loop
- while loop is more versatile but can be a common source for errors
- sometimes, programmers can get stuck in an infinite loop/ does not stop
  ```

    - can get out of the infinite loop by pressing control c
    - interactive loop/ allows users to repeat certain portions of the program on demand
    - a sentinel loop / continues to process data until reaching a special value that signals the end
    - any value can be chosen as a sentinel value
    - the sentinel is not processed as part of the data
    - priming read / avoiding sentinels
- loop test (while statement) proves that the item is not sentinel before processing it
- when the sentinel is reached, the loop terminates
- the sentinel loop is a very handy pattern for solving all sorts of data processing problems
- one simple solution for a sentinel is to have it be the value of an empty string
- when a user hits enter, this returns an empty line which indicates stopping of code if empty string is
  sentinel
- Boolean operators: and, or, not
- Boolean operators and / or used to combine two Boolean expressions and produce a Boolean result
- the and of two expressions is true when both the expressions are true,
    - and = true when both expressions are true
- the or of two expressions is true when either expression is true
- only time or is false when it is false and false
- not operator computes the opposite of a Boolean expression
- not operator is an unary operator / meaning that it operates on a single expression

- if an expression is true, it is false. if an expression is false, it is true
- needs to be evaluated in a specific order: not>and > or
- boolean expressions obey Boolean logic/ boolean algebra
- anything ored with true is just true
- both and and or distrubute over each other
- ex. a or (b and c) == (a or b) and (a or c)
- double negative cancels out
- ex . not(not a) == a
- DeMorgan's law
  not(a or b)== (not a) and (not b)

  not(a and b) == (not a) or (not b)

- input validation / validates user inputs and prompts for another input if incorrect
- post test loop = condition tests comes after the loop body / must always execute the body of the loop at least once
- break statement/ used to leave a infinite loop
- loop and a half / when the loop exit is in the middle of the body
- Boolean operators are short circuit operators / this means

operator

operational definition

x and y —— if x is false, return x. otherwise, return y

x or y —— if x is true, return x. Otherwise, return y

not x —— if x is false, return true. otherwise return false

that a true or false value is returned and in an or where the first expression is true , python will not even