|  |  |  |
| --- | --- | --- |
| Testing definition | | Finding the problem |
| Debugging definition | | Fixing the problem |
| Is testing & debugging one step? | | No, 2 separate steps |
| Testing should be a formal process. T/F | | T |
| Test case includes: | | 1) Program being tested  2) input  3) written statement of what is expected to happe |
| When is a test case considered successful? | | When the program being tested does what test case anticipated |
| Testing should include valid input data & invalid input data. T/F | | T |
| Testing may show presence of bugs, but never their absence. What does this mean? | | We could test forever, but even if we don’t find bugs, it doesn’t mean there aren’t any |
| Testing is science, but not art. T/F | | F |
| Testing should be done last after the program is completely written. T/F | | F; Leads to very buggy code |
| Computers make mistakes. T/F | | F; NEVER |
| Failure definition | | erroneous behavior of system that produce unexpected results |
| Error definition | | The amount by which a failure is incorrect |
| Fault definition | | something wrong in the software which MAY produce a failure |
| Mistake defintion | | Human action which results in software w/t a fault |
| Mistakes are entirely human activity. T/F | | True |
| When a human makes a mistake, what happens? | | It leaves a fault in the code |
| A fault in the program always leads to failure. T/F | | False; It MAY result in a failure |
| Can the amt of error be determined after a failure occurs? | | Yes |
| Testing another definition | | process of exercising a system to produce failures |
| Debugging another definition | | process of locating, analyzing, and correcting suspected faults |
| Why do we do testing? | | To produce a failure |
| Testing doesn’t show absence of bugs, so what do we do? | | We have a test plan, execute it, keep record of the results |
| A well-designed test plan is likely to uncover the most expected faults. T/F | | T |
| Test case definition | | Specific set of data that will be entered as input and result we expect to see from the system |
| Some test cases intended to produce a failure and that’s considered a successful test. T/F | | T |
| Black Box Testing | | Program inside a black box that we cannot see inside of. Don’t know how the program does what it does, just that it is supposed to do it. We have SPECIFICATIONS, wo we know what the program is supposed to do |
| What kind of test cases in black box testing ? | | solely set of inputs and the output we expect to get back |
| Clear Box Testing | | The program is an open window that we can see inside of . Look at individual lines of code and choose test case data based on what we see. |
| What kind of test cases in clear box ? | | If there are bunch of if, elif, elif, else statements, data should be tried using every branch of if statement meaning lot of test cases |
| If there is a loop in clear box testing , what do you do? | | execute the loop 0 times, one time, more than one time, and too many times . Executing 0 times the loop means it doesn’t run at all |
| If there is a recursion in clear box testing , what do you do? | | Create test cases so data will not cause recursive call to happen and other cases where recursive call happens |
| What’s another name for clear box testing? | | White box testing |
| Flow path testing | | Draw flow of program as it moves from structure to structure and apply math graph theory to see what path should be tested |
| partition testing | | Apply math set theory to input data & find equivalence classes that form partition of that set and include test cases from each equivalence class in the set |
| boundary value testing | | choose input data on each side of any boundaries |
| Testing mindset extends to designing a program. T/F | | T |
| Data validation | | block flawed data at point of entry   * isdigit() tests a string to see if all contents are numberals 0 thru 9 * non-numerals, then while loop repeats * How do u check for a real valued number entered? |
| What is the difference b/t black box testing & clear box testing? | |  |
| How is it true that a computer cannot make a mistake? | |  |
| What comprises a test case in a test plan for a given program? | |  |
| Why is it that we don’t just test until we get all the problems out? | |  |
| bugbear | | Webster’s says something that is a ‘source of irritation” or a problem |
| Hawkin’s New Catechism of Electricity (1896) | | bug is used to designate fault or trouble in electric apparatus |
| Grace Hopper coined the term bug in relation to faults in computers. T/F | | F; She found moth while working on Harvard Mark II and she taped it to log book. It’s just a tale – not real |
| Computer definition in the olden days | | Human mathematicians such as Lt. Hopper |
| Harvard Mark 1 & ENIAC relationship | | 1944 vs 1945 contemporaries of one another at University of California |
| coders would write the program, what would programmers do? | | load it onto those huge machines |
| First fully programmable | | Mark 1 |
| First fully electronic | | ENIAC |
| First computer by court decree | | Atanasoff-Berry Computer built in Iowa State University in 1942 but it was not Turing-complete |
| Code breaking Colossus computer was known to be working before many other computers including Atanasoff-Berry computer. T/F | | T |
| Every non-trivial program has no bugs in it. T/F | | F |
| The number of bugs in a program has no relation to the number of bugs that have already been removed from that program. T/F | | T |
| When do you start debugging? | | * Completed writing the program * Tested the program and exercised the code until failure occurred * Failure indication of fault in code, so time to fix the fault |
| Bugs can be described in different ways: | |  |
| Covert bug | | Might produce failure but only very rarely. No obvious manifestation |
| Overt bug | | It causes program to crash or do something wrong and is very easily detected. It is so obvious |
| Intermittent bug | | Seems to be present but then not there at times. It takes particular set of input data to trigger it |
| Persistent bug | | Occurs every time the program is run with the same inputs |
| What kind of bugs do we want? | | Overt and persistent bugs |
| The key to being a good debugger is being systematic in conducting your search. T/F | | T |
| Debugging and relation to science and art | | plan and conduct experiments to find bugs |
| First step of debugging | | locate the line of code that contains the fault. Occurs when you observe a failure. Failure means there is a fault. Can’t locate exact line containing the fault then narrow it to couple of lines. |
| Second step of debugging | | Know where the fault is, now find out what the fault is. Develop hypothesis, create set of data to test hypothesis , decide beforehand what the outcome of your test data is and write it down |
| It’s hard to prepare data that shows that hypothesis is wrong. T/F | | F |
| Third step of debugging | | Conduct the experiment, run the test data, collect data |
| In third step of debugging, if your data cannot disprove hypothesis, then what does this mean? | | You found out what is wrong and tighten hypothesis to get even more fine-grained data |
| Usual suspect bugs | | * off by one error(obo)   range (1,10) or range (0,10)   * equality = vs assignment ==   (doesn’t happen much in Python)   * Failed to initialize variable   sum = 0   * Passed wrong variable instead   Keep formal & actual parameter names separate   * Misspelled variable   insidious in Python( case-sensitive)   * Passed parameters in wrong order * Tested for equality too tightly   num<1 better than num==  especially for floating point values   * Tested equality rather than objects   L1==L2 diffres from id(L1) == id(L2)   * Mixed up punctuation * Is there an alias somewhere?   Could othere code be changing a value? |
| When you get a headache while debugging, what do you do? | | * Have someone else look at the code * Explain problem to someone else * Looking at wrong line of code that you think has fault, might be elsewhere * Walk away and do something else and then come back to the problem |
| Verification vs Validation | | Did we build the system right? checks to see if specification done correctly vs Did we build the right system ? confirms the program as written ,meets the specification |
| Testing and debugging are part of validation . T/F | | F; verification |
| What should you do right after finishing testing? | | Confirm the specification is correct |
| What is the difference b/t testing and debugging? | |  |
| We can’t just test until all the bugs are found, so what do we do? | |  |
| How is it that debugging really makes computer programmers into scientists? | |  |
| Why is it so helpful to get someone else to help you debug a program? | |  |
|  | | Either error of syntax (not spelled correctly) or semantic error (something doesn’t mean what we want it to mean) |
|  | |  |
| exception false definition | | something that does not conform or something unexpected . Misnomer b/c we want to try to expect everything unexpected. |
| exception true definition | | some condition or situation outside of normal |
| raising an exception | | signaling that an exception has occurred |
| handling an exception | | responding to an exception that has been signaled, to get the program back on track |
| exceptions come in two ways: | | built in exception- defined by python for things that commonly go wrong  user defined exception- program writing has own unique problems |
| Python does exceptions with try statement: | |  |
| Try statement, when executed what happens? | | All code inside try statement doesn’t  cause harm, then exception clause ignored. But if try code causes exception to occur then exception handler matching that name is executed. Exception clause is like a conditional statement |
| If try block of code causes exception to occur, the exception handler matching that name is executed. Then what happens? | | If there is handler that matches the exception, the situation cleaned up and program goes on and it picks up after try statement |
| If try block of code causes exception to occur, but no matching handler. Then what? | | Exception is unhandled and the program crashes just like before |
|  | | Exception handling covers a broader range of possibilities than data validation |
| Parameterize exception handling to deal with input | |  |
| Exception catches mistake. If you want something to repeat, then what? | | Put it inside iterative loop |
| Polymorphism | | Ability of function to handle arguments of various data types. Crucial to understanding oop. |
|  | | * Can have multiple exception clauses as you like * They are processed in order |
|  | | * Each exception clause has an exception name * If so, only that exception is handled by that clause * You can omit exception name * In this case, any unhandled exception would be caught by that clause * All built-in exceptions will be automatically raised when circumstances warrant * You can manually raise an exception you want |
| What exceptions are there? | | Python library or play with python. Angry red text, look up exceptions |
| Programmers do not have a built in pseudorandom number generator. T/F | | False, They do. Pseudorandom means numbers are not really random. Anyone who attempts to generate random numbers by deterministic means is living in a state of sin - Neumann |
| Pseudorandom number generator- How does it work? | | Algorithm generates digits in a fashion that appears to be random. Random number generator takes an argument called a seed to get it started. Using the same seed will create exact same sequence of numbers. Useful in simulations when you want to repeat a “random” number sequence |
| Python’s random generation comes in module named | | random.py must have from random import \* in code |
| How do you initialize a seed? | | random.seed(9001)   * you seed the generator just once, then you can get random numbers repeatedly out of it * every time you use same seed, you WILL get SAME numbers |
|  | | What happens when a different seed is used? The results change |
| We normally seed only once. T/F | T | |
| We want numbers that are irreproducible. If we write program that plays game, we want random things to appear random. How to do this in Python? | Is easy in Python – don’t seed random number generator at all then. Leaving step out causes Python to get random number generated by Operating System | |
| Benefit of getting random number generated by operating system? | Portability – your program would work fine even if it’s on a different machine | |
| The most random way to do program? | seed generator with system clock or get random numbers from random.org | |
| Want random number drawn from some range of integers: |  | |
| Program to write file filled with random integers : |  | |
| Program continued |  | |
| What is an exception? |  | |
| What is the difference b/t raising an exception and handling an exception? |  | |
| How is Python’s exception mechanism sort of like an if-elif-else statement? How is it different? |  | |
| What is the tip to make a random number generator spit out the same random numbers in a row, every time? |  | |
| What’s Python’s way of making them as random as possible? |  | |
| assertion | claim you make inside that program that will be evaluated when that line of code is reached in the program | |
| If assertion evaluates as true: | program goes on | |
| If assertion evaluates as false: | Python raises an AssertionError | |
| What’s the point of assertions? | Good way to poke into what’s going inside a running program. Useful for debugging. Unlike troubleshooting print() you can leave assertions in code for future debugging | |
|  |  | |
| How are assertions useful in making code safer? | * You can put assertion to confirm data type of an argument passed to function. Pyton doesn’t type check arguments, so overcomes this * You can put in assertions to confirm invariants haven’t changed * In developing large systems, good use of assertions can offset Python’s lack of type safety | |
| When should assertion checking be disabled? | If run time speed is more important than type safety | |
| Python is not in step with other contemporary programming languages when it comes to assertions. T/F | F | |
| Classes and object-oriented programming relationship to Python | It’s built into Python | |
| ADTs how long to figure out? | 40 yrs for Computer Science to figure out | |
| Abstraction | overlooking/hide away certain details for the moment and pull them out when we need them | |
| 2 kinds of Abstraction: | Data Abstraction  Process Abstraction | |
| earliest machines used process abstraction. T/F | F; used both abstractions | |
| Data abstraction is natural. T/F | T | |
| You can’t put “7” or” 3.14” or “Please enter a value “inside of a computer | binary numbers and text codes used for internal representation | |
| Which one came first? Data abstraction or Process abstraction? | Earliest machines used both – process abstraction came along immediately with data abstraction. Routines were done over and over, notebooks were kept so they didn’t have to reinvent code | |
| FORTRAN | introduced implicit data typing based on variable name used   * said all variables with names staring with letters L-N would be integers and everything was a real number * “God is real, unless declared as an integer * integers and values need different forms of memory allocation | |
| 1960s & 1970s programming language design focused on data types | APL & PLI languages came out with built-in data types | |
| data structure | stores data for a program | |
| data flow diagrams | helps you understand what a program is needed to do | |
| strong data typing relationship to programs | make very safe programs | |
| 198s-1990s concept | data abstraction + process abstraction together into one unit. Concept now is OOP | |
| Key aspect of OOP | encapsulation- put things inside box and keep all of it together. Limits the access other things have to the things inside the box | |
| interface | connection b/t what’s inside the box and what outside entities can do | |
| one container | contains the storage places for data we need , tools to manipulate data, and connecting interface so outside code can use it | |
|  | array, public operation bridge to private operation , interface | |
| array | inside each object we create the data storage needed to hold info whether it be variables or structured types | |
| public operation bridge to private operation | we provide tools to manipulate internal data, report it, and modify it | |
| interface | describes how outside code will interact with the object | |
| data stores and methods are encapsulated meaning? | Protected from outside reach, except through interface | |
| program with fractions using ADT | * check if fractions are same value * define mathematical operations * prevent certain fraction value from ever occurring | |
|  | create a new Fraction object, how to find num/denom, how to multiply fractions, how to return new fraction | |
|  | how to divide fractions and how to print out as string | |
| Python implements abstract data type in mechanism | class | |
| class has two data members in fraction program | numerator and denominator stored internally as integers where integer is a data type | |
| data type | set of values together with set of operations that | |
| abstract data type | data type together with methods used with that data type | |
| Methods of ADT | recursively like operators of basic data type | |
| create class | creating new data type specifically designed to our needs for that program | |
| class methods for each class you create includes | constructor- create new objects of that class  method to output object as string  getter methods to output each of the data members of the class  setter methods to update the values of each of data members of the class | |
| Is it okay to omit Setter methods ? | yes for desired safety | |
| class cannot be a data type. T/F | false | |
| each object | instance of that class | |
| instantiating the object | creating a new object | |
| second key aspect of oop | inheritance | |
| inheritance | class to be defined as special case of an already existing class | |
| special case | inherits features of earlier class | |
| new class | child class /subclass | |
| parent class | new class inherited from this class/ superclass | |
| Create person class | each instance of class has name, birthday.  name is set at instantatination  setter method set bday  age worked out from bday but not stored | |
|  | * subclass created * each MITPerson has all attributes of Person object plus new one * class data member IDNum has been added | |
| inheritance can go indefinitely | * Particular kind of MITPerson would be student. With this new class, no additional attributes were added * 2 special cases of students: UG subclass w/t additonal attributes and Grad subclass with no additional attributes | |
| “Programming is managing complexity to facilitate change. There are 2 powerful mechanisms available for accomplishing this: decomposition and abstraction. Decomposition creates structure in program and abstraction suppresses detail. The key is suppress the details.” | MAPS breakdown step includes decomposition   * We simply can’t handle entire problem all at once * So we break it down into pieces, and work one piece at a time * Provides for modularity and flexibility in program’s design | |
| Abstraction in class inheritance approach | * UG class- we don’t have to worry about birthday and ID numbers * members of the class have those though * We didn’t have to focus on thinking about such details that were already handled * inheriting from an existing class took care of that for us | |
|  |  | |