Solidity files…

a) Can't be split across multiple files, everything should be in one single file
b) Can be split across multiple files, but every contract must be in a file with the same name as the contract itself
c) Can be spread across multiple files. To import all contract from a file you can use "import 'myfile.sol'. To import Contract MyContract from myfile.sol you use "import {MyContract as SomeContract} from 'myfile.sol';".

Files can be…

a) Imported using relative and absolute paths, where the "." And the ".." depict that it's a relative path.
b) Imported only via GitHub using the Repository and Username.
c) Imported using the special requirefile(…) statement, which looks in a specific library path to import files.

Importing from GitHub…

a) Works across all compilers and platforms the same way
b) Is generally possible, but currently works only in Remix, but doesn't work in Truffle

Single line comments in Solidity are working with

a) Either // or ///
b) With /* comment */ or /** @.. natspec style */
c) Are not possible, all comments must be multi-line

Multi-Line Comments in Solidity work with

a) Either // or ///
b) With /* comment */ or / @.. natspec style */**
a) Are not possible, all comments must be multi-line

The following are value types in Solidity

a) Integer, Boolean, Struct, Mapping and Enum
b) Integer, Boolean, Enum and Addresses
c) Integer, Boolean, Structs and Fixed Point Numbers

To compare a String in Solidity you use

a) String1 == string2
b) The internal function "str_compare(str1,str2)"
c) You can't directly compare two strings, but one method would be to hash both strings and compare the hashes
d) Bytes32(string1) == bytes32(string2)

If we divide two integers: 5/2, the result is

a) 2, because the decimal is truncated
b) 3, because it's always rounded
c) 2.5, because it's automatically converted into a float.

A Struct is a great way

a) To define a new datatype in Solidity, so you don't need to use objects of another contract
b) To hold instances of other contracts
c) To implement pointers to other contracts that can hold new datatypes

A Mapping consists of keys and value.

a) The Keys can be anything, but the value can't be another mapping or struct
b) The Value can be anything, but the key cannot be another mapping, struct, integer or Boolean
c) The value can be anything, but the key cannot be another mapping, struct, enum or dynamically sized array

To Iterate through a Mapping you

a) Can use the length parameter of the mapping
b) You need an external helper variable
c) You cannot iterate any mapping to make the overall language design more safe

Function and Variable Visibility:

a) A function marked as internal cannot be called by other contracts, unless the function is used by a derived contract. Private Functions cannot be called by any other outside contract and public variables are generating automatically a getter function.
b) A function that is marked as external can never be called internally. Private functions can also be called by derived contracts using inheritance. Private variables are accessible also in derived contracts.

View and Pure Functions:

a) A function marked as pure can change the state, while a view function can only return static calls
b) A function marked as view can never access state variables, while pure functions are here to return only one value
c) A view function can access state variables, but not write to them. A Pure function cannot modify or read from state.

View and Pure Functions

a) Can only be accessed during calls
b) Can be accessed during transactions and calls

The Fallback function

a) Cannot receive Ether, not even by adding the payable modifier
b) Can contain as much logic as you want, but it's better to keep it short and not exceed the gas stipend of 2300 gas
c) Can be used to avoid receiving ether.

To get the address that initiated the transaction you need to use

a) Tx.origin
b) Msg.sender

If a User calls contract A and that calls Contract B, then msg.sender in Contract B will contain the address of

a) The User
b) Contract A

Loops in Solidity

a) Are a great way to circumvent gas requirements, because a loop will only consume gas once
b) Are dangerous when used with datastructures that grow, such as arrays or mapping, because it's hard to estimate the gas requirements
c) Should be avoided where possible, because of unknown side-effects on the gas requirements

Events

a) Are stored on chain and are a great way to get a return value when a contract calls another contract
b) Are stored in something like a side-chain and cannot be accessed by contracts
c) Are used primarily for debugging exceptions in solidity

According to the official Style Guide

a) You should capitalize function names, events and contract names, to avoid confusion with JavaScript. You should use Tabs to indentation and a maximum of 80 characters per line.
b) Contract names should be capitalized, while functions should be mixedCase. You should use 4 spaces as indentation and a maximum of 79 (or 99) characters per line.
c) Contract should be mixedCase, as well as function names. Events should be capitalized. 2 spaces should be used as indentation and a maximum of 120 characters per line.

A version pragma is a great way

a) To make it clear for which compiler version a smart contract was developed for. It helps to avoid breaking changes.
b) To make it clear for which blockchain a smart contract was developed for. It helps to avoid confusion with beta-customers.
c) To make it clear for which blockchain node a smart contract was developed for. It helps to avoid mixing up different versions of go-ethereum.

Variables of the type address store

a) A 20 bytes value
b) A 32 bytes value
c) A string
d) A 20 characters long hex number