55. XSL се използва за **– трансформиране на XML документ към друг текстов документ**

56. Разгледайте <xsl:value-of> елемента. Ако стойността на неговия select атрибут е select=‘.‘ тогава ние избираме – **текстовия контекст на елемента и текстовия контекст на всички наследници на елемента**

57. Изпълнението на XSLT декларациите <xsl:value-of select=“.“/> и <xsl:value-of select=“text()“/> води – **до един и същ или до различни резултати в зависимост от типа на съдържанието на текущия елемент**

58. В XSLT, вземането на решение кои елементи ще бъдат обработени се задава със следния XSLT елемент - **<xsl:apply-templates>**

59. XSL елементът Apply-Templates се използва вътре в един шаблон (template) за извикване на други шаблони. Той **– активира рекурсивно обработката на всички наследници на елемента, за който се отнася**

60. При избиране на елемент, наречен MyElem и имащ атрибут Attr със стойност title, в XSLT ние трябва да използваме – **select=“MyElem[@Attr=‘title‘]“**

61. Ако сме дефинирали XSL променлива като <xsl:variable name=“price“>low</xsl:variable>, то тя може да се използва в XSL елемент като - **<xsl:value-of select=“$price“/>**

62. При прилагане на XSLT трансформацията

<xsl:template match=“name“>

<xsl:element name=“{.}“>

Very nice!

</xsl:element>

</xsl:template>

За документа

<names>

<name>Bob</name>

<name>Steve</name>

</names>

имената на създадените елементи в резултатното дърво ще бъдат – **със съдържанието на елементите в изходящото дърво**

63. XLS кодът, показан на фигурата, ще даде като резултат:

<xsl:template match=“nam

<xsl:element name=“{

My own contents!

</xsl:element>

</xsl:template>

Изходните XML елементи ще бъдат с имена, еднакви със съдържанието на елементите от сорс дървото, и със съдържание „My own contents!”

//май снимката е отрязана отдясно и това не е пълния код

64. XPath изразът //book[@pages] връща – **всички book елементи, които имат атрибут pages**

65. XPath изразът ./book[author/last=“пробен изпит“] връща – **елемент book – наследник на текущия елемент, който има елемент author с поделемент last равен на „пробен изпит“**

66. При използването на XPath text() функцията, ние избираме **– текстовия контекст само на елемента**

67. Приложения, които имат нужда от сложни манипулации на много от XML елементите, трябва да използват – **DOM**

68. С един DOM Element обект – **не може да направите разлика между подразбираща се (default) стойност, определена в DTD, и стойността, дадена в XML файла**

69. Кое от твърденията е истина – **само DOM Element обектите имат атрибути**

70. Всеки DOM възел (Node) може да има деца – **невярно**

71. Методът getAttributes() на DOM интерфейса Node връща – **NamedNodeMap**

72. DocumentType::Node Interface се използва за получаване на информация за документ, описан в DTD – **DOM 1.0 не разрешава редактиране на този възел**

73.DOMException връща HIERARCHY\_REQUEST\_ERR при опит за – **вмъкване на възел на неподходящо място в йерархията на DOM дървото**

74. SAX служи както за четене на XML документи, така и за генериране на XML – **Невярно**

75. XMLReader в SAX 2.0 разширява стандартния Java Reader интерфейс – **Невярно**

76. В SAX приложните обекти, имащи достъп до XML сорса – **трябва да бъдат регистрирани (от програмиста) за callback функциите на парсера**

77. SAX Element обектите – **могат да разграничават атрибутите, дефинирани изрично, от тези специфицирани в DTD**

78. Събитието processingInstruction възниква, когато SAX парсерът достигне до всяка една инструкция за обработка, включително и до XML декларацията - **Невярно**

79. Изберете едно – **SAX използва push парсване, а StAX – парсване от тип pull**

80. При парсване на XML документи посредством StAX, можем за се придвижваме само напред в XML документа – **Истина**

81. За постигане на по-малък, ефикасен и бърз код с използване на StAX, се препоръчва – **cursor API**