SWINBURNE UNIVERSITY OF TECHNOLOGY

Introduction to Programming

Week 6: **Data, References, Arrays and Searching**

KNOWING

Overview

- Character data types
- Arrays, lists and searching
- Using references in Ruby
- Designing the Text Music Player
- Tests

ASCII

- ASCII (American Standard Code for Information Interchange). Used since 1960.
- It represents the 128 characters as found on standard keyboards.
- It uses only 7 of the 8 bits (the first bit is always zero).

Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8
Null	2	4	8	16	32	64	128

ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

UTF-8

- But there are many more characters to be represented if we want to represent a wider set of characters (eg: Greek, Arabic, Chinese, Korean etc).
- To represent a more complete character set the Unicode standard was developed (in 1991).
- UTF-8 is one implementation of the Unicode standard. It is backwards compatible with ASCII (for non-ASCII the first bit is non-zero).

UTF Codes

U+2713: CHECK MARK

Tweet

Your Browser	✓
Index	U+2713 (10003)
Class	Other Symbol (So)
Block	Dingbats
Java Escape	"\u2713"
Javascript Escape	"\u2713"
Python Escape	u"\u2713"
HTML Escapes	✓ ✓
URL Encoded	q=%E2%9C%93
UTF8	e2 9c 93
UTF16	2713

U+0394: GREEK CAPITAL LETTER DELTA

Tweet

Your Browser	Δ
Index	U+0394 (916)
Class	Uppercase Letter (Lu)
Block	Greek and Coptic
Java Escape	"\u0394"
Javascript Escape	"\u0394"
Python Escape	u"\u0394"
HTML Escapes	Δ
URL Encoded	q=%CE%94
UTF8	ce 94
UTF16	0394

```
#source: https://www.charbase.com

checkmark = "\u2713"
puts checkmark.encode('utf-8')

omega = "\u0394"
puts omega.encode('utf-8')
```

KNOW ING

Using Dynamic Arrays

- Remember we looked at Dynamic Arrays:

```
def main
  name_array = Array.new()

  count = read_integer("How many names: ")
  index = 0
  while (index < count)
    name_array << read_string("Enter next name: ")
    index += 1 # Increment index by one
  end

  index = 0
  while (index < count)
    puts name_array[index]
    index += 1 # Increment index by one
  end
end
```

Lists

- Lists are another type of data structure. We look at lists as having a **head** and a **tail**.
- eg: ["apple", "pear", "orange"]
- the head is the element "apple, the tail is the list ["pair, "orange"]
- In Ruby Arrays can be treated as lists.

Fun Fact

- In some languages (Lisp, Scheme) a different terminology is used: **car** and **cdr**.
- car = “Contents of the Address part of the Register”
  cdr = “Contents of the Decrement part of the Register” (“could-er”)

- From the 1950s inplementation of LISP on an IBM 704.

List Operations

An array is a data structure - it simply is an organisation of memory. But a list generally comes with a set of additional operations that are useful for using data as a list.

A list is a way of viewing and using data.

List data structures typically allow the following operations:

- items can added to them,
- you can take the head (the first item)
- you can ask if a list includes an item
- you can apply an operation to each item in the list
- You can insert items at a particular point

These are not things that languages (such as C) provide automatically with arrays.

Basic List Operations

Here are some of the simple operations you might do with lists:

```
mylist = ["apple", "pear", "orange"]

puts "Original list: "
puts mylist

mylist += ["banana", "plum"]

puts "List with another list ['banana', 'plum'] added: "
puts mylist

mylist.insert(3, 'cherry')
puts "List with an item 'cherry' inserted after position 3: "
puts mylist
puts mylist
```

Complex List Operations - reject

```
list = [2, 4, 6, 8]
puts "Original list: "
puts list

newlist = list.reject do |i|
  if i > 2
    true
  else
    false
  end
end

puts("New list is: ")
puts newlist
```

Complex List Operations - each

- This second one applies an operation to each item in the list if the item matches a condition - it:
- Adds one to each item in the list and returns the changed items as a new list
- Goes through the list printing out each element and its position

```
list =[2, 4, 6, 8]

newlist = list.collect do |i|
  i + 1
end

j = 0
newlist.each do |i|
  j += 1
  puts "Item #{j} is #{i}"
end
```

Searching an Array

- So we have seen some complex list operations performed on arrays.
- One thing we often want to do is to see if an array contains an item.
- To do this we need to search through the array and check each item to see if it matches the item we want.
- If the item is there, we then want to be able to get it, so we might need its position.
- We can do this as follows using a while loop:

```
# this returns the index of the item or -1 if it is not found
def search_array(a, search_item)
  index = 0
  found_index = -1
  while (index < a.length)
    if (a[index] == search_item)
      found_index = index
    end
    index += 1
  end
  return found_index
end

def main
  my_array = ["apple", "pear", "banana", "orange"]
  result = search_array(my_array, "orange")
  puts "The index of the item searched for is #{result}"
end

main
```

Using references in Ruby - Lights example

- Ok, now we will run the Lights example, which will allow us to look at pass-by-value versus pass-by-reference and also explore further the use of GOSU.

Designing and building the Text Music player

- Lets us look at:
- Bottom up design for the Text Music player task
- The structure chart for that task
- Writing the code and testing it for that task.

Tests

- We look at the two tests:

Test 1

- This is run in workshops in Week 8 – watch for announcements.
- **Test 2**
- This is run in the workshops in Week 10 - watch for announcements.