CAML

10 - Autres types et structures

http://tsi.tuxfamily.org/OCaml

4 juin 2024

OCaml distingue :

- Le caractère 'a'
- La chaîne de caractères "a"

```
# let x = 'a';;
val x : char = 'a'
# let x = "a";;
val x : string = "a"
```

Différences avec Python

Opérations sur les chaînes : on utilise le module String d'où la syntaxe (proche de Python) :

- Concaténation : ^

```
# "bon" ^ "jour";;
- : string = "bonjour"
```

- Connaître la longueur d'une chaîne :

```
# String.length "OCaml";;
- : int = 5
```

- Connaitre le caractère d'indice *i* :

```
# let txt = "OCaml" in txt.[1];;
- : char = 'C'
```

Une chaîne est une donnée **non mutable**. Depuis la version 4.02 d'OCaml, on ne peut plus modifier un élément d'une chaîne (comme en Python).

D'autres fonctions peuvent-être utiles pour vos TIPE, mais n'ont pas à être connues :

- Créer une chaîne de longueur donnée :

```
# String.make 3 'a';;
- : string = "aaa"
```

(Le second argument est un caractère et non une chaîne).

- Extraire une sous-chaîne :

```
#let txt = "OCaml est un langage fonctionnel"
      in String.sub  txt 13 7;;
- : string = "langage"
```

Plus de fonctions sur la documentation officielle :
https://v2.ocaml.org/api/String.html
https://v2.ocaml.org/api/Char.html

- Obtenir le code ASCII d'un caractère et inversement :

```
#int_of_char `A`;;
- : int = 65
#char_of_int 70;;
- : char = `F`
```

American **S**tandard **C**ode for **I**nformation **I**nterchange
Code américain normalisé pour l'échange d'information

x10 \ x1	0	1	2	3	4	5	6	7	8	9
3			espace	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	□	€	

Conversions :

- `int_of_string "100"` renvoie 100
- `float_of_string "100"` renvoie 100.0
- `string_of_int 100` renvoie "100"
- `string_of_float 100.0` renvoie "100.0"

Plus généralement, `type1_of_type2` convertit un élément de type2 en un élément de type1.

Les chaînes de caractères ne sont pas mutable, néanmoins, un autre type (**Bytes**) représentant un tableau **mutable** d'octets de longueur fixée permet de contourner le problème si besoin. Les octets sont donnés sous forme de caractères en correspondance avec leurs codes ASCII.

```
# let txt = Bytes.of_string "CAML";;
val txt : bytes = Bytes.of_string "CAML"
# Bytes.get txt 1;;
- : char = 'A'
# Bytes.set txt 3 'E';;
- : unit = ()
# Bytes.to_string txt;;
- : string = "CAME"
```

Ce type n'est pas à connaitre, plus d'infos :
https://v2.ocaml.org/api/Bytes.html

Au moment de la conception du programme, on est amené à définir des types abstraits en spécifiant quelles propriétés ils doivent vérifier (spécification fonctionnelle)

- Le développeur du type devra respecter ces propriétés, mais sera libre de l'implémentation concrète
- L'utilisateur pourra supposer que les propriétés sont vérifiées pour son code

Vocabulaire

les **dictionnaires** (aussi appelés **tableaux associatifs** ou **tables d'association**).
On veut pouvoir associer des clefs $k \in \mathcal{K}$ à des valeurs $v \in \mathcal{V}$ et qu'il soit possible

- de créer un dictionnaire vide
- d'insérer une nouvelle association
- de rechercher à quelle valeur est associée une clef
- de supprimer une association
- tester la présence d'une clef

En général, $\mathcal{K}$ est supposé totalement ordonné. Dans la suite, on supposera qu'à une clef n'est associée qu'une seule valeur au maximum (le dictionnaire est une fonction au sens mathématique)

- `type ('k,'v) dict`
- `creer : unit() -> dict`
- `inserer : dict -> 'k -> 'v -> dict`
- `rechercher : dict -> 'k -> 'v`
- `supprimer : dict -> 'k -> dict`

On ajoute souvent la fonction : `present : dict -> 'k -> bool`

- une recherche dans un dictionnaire vide renvoie une erreur.
- une recherche de la clef k dans le dictionnaire où l'on vient d'ajouter l'association $k \mapsto v$ renvoie v.
- une recherche de la clef k dans le dictionnaire où l'on vient d'ajouter l'association $k' \mapsto v$ avec $k \neq k'$ renvoie la même valeur que la recherche avant l'ajout.
- une recherche de la clef k dans le dictionnaire où l'on vient de retirer l'association à k renvoie une erreur.
- une recherche de la clef k dans le dictionnaire où l'on vient de retirer l'association à k' avec $k \neq k'$ renvoie la même valeur que la recherche avant la suppression.

Différentes implémentations possibles
Avec une liste : voir TD

A l'aide d'un arbre binaire de recherche :

A l'aide d'un arbre binaire de recherche :

```
type ('k,'v) dict =
  Nil
 |Noeud of ('k,'v)dict * ('k * 'v) * ('k,'v)dict
```

A l'aide d'un arbre binaire de recherche :

```
type ('k,'v) dict =
  Nil
 |Noeud of ('k,'v)dict * ('k * 'v) * ('k,'v)dict
```

```
let creer () = Nil
```

A l'aide d'un arbre binaire de recherche :

```
type ('k,'v) dict =
  Nil
 |Noeud of ('k,'v)dict * ('k * 'v) * ('k,'v)dict
```

```
let creer () = Nil
```

```
let rec rechercher d k = match d with
      Nil -> raise Not_found
    | Noeud(_, (k',v), _) when k = k' -> v
    | Noeud(fg,(k',_), _) when k < k'
                          -> rechercher fg k
    | Noeud(_, (k',_),fd) -> rechercher fd k
```

```
let rec inserer d k v = match d with
    Nil -> Noeud(Nil,(k,v),Nil)
  | Noeud(_, (k', _), _) when k = k'
              -> failwith "cas non traité"
  | Noeud(fg,(k',v'),fd) when k < k'
              -> Noeud(inserer fg k v,(k',v'),fd)
  | Noeud(fg,(k',v'),fd)
              -> Noeud(fg,(k',v'),inserer fd k v)
```

A l'aide de tables de hachage.

Le module `Hashtbl` permet la création de dictionnaires mutables (les fonctions seront rappelées en concours) :

- `Hashtbl.create : int -> ('a, 'b) Hashtbl.t`
- `Hashtbl.add : ('a, 'b) Hashtbl.t -> 'a -> 'b -> unit`
- `Hashtbl.remove : ('a, 'b) Hashtbl.t -> 'a -> unit`
- `Hashtbl.mem : ('a, 'b) Hashtbl.t -> 'a -> bool`
- `Hashtbl.find : ('a, 'b) Hashtbl.t -> 'a -> 'b`