**Résumé PYTHON**

**Mémoire :** espace virtuel à taille finie dans lequel l’on stocke des données

Adresse : Chaque emplacement en mémoire possède un identifiant unique nommé adresse. Chaque adresse à un emplacement de 1 bit. Plan d’adressage indexé selon des octets.

**Indexation :** méthode d’organisation et de répartition des données en mémoire s’appelle plan d’adressage.

**Mem Vive Random Access Memory :** composant électroniquement proche du processeur/accès rapide / Temps de vie limité / Mémoire cache

**Mem Morte Read Only Memory :** Zone de stockage etendue / éloignée du processeur / Temps de vie illimité

Lois de Boole + Lois de Morgan

**La compilation :** est l’étape consistant à traduire le code source en langage machine. On génère un fichier binaire qui sera exécuté par la machine. Le compilateur vérifie la validité syntaxique et grammaticale du code. Si la syntaxe est incorrecte, il renvoie des erreurs et stoppe la compilation.

**Théorie de l’information :** Étude des modèles de production et de transmission d’information via un canal de communication donné. Étude probabiliste et statistique de l’évolution de l’information dans un système de communication

**Théorie de graphe :** Étude de graphes et modèles pour la résolution mathématique et algorithmique de problème fondamentaux.

**Calculabilité :** Vérifier la possibilité d’évaluer une fonction par un algorithme en un temps fini. Démontrer que l’on peut ou non calculer la valeur d’une fonction f en tous points.

**Théorie des langages :** Étudier la structure, syntaxe et grammaire des langages de programmations

**Von Neumann :** architecture logicielle permettant de représenter sommairement le comportement d’un ordinateur. Un ordinateur qui utilise l’architecture de Von Neumann, les programmes et données existent dans une seule forme de mémoire.

Architecture fondamentale | Séparation contrôle/calcul | Instructions et données en mémoire | Un programme = une donnée | Pas de protection de la mémoire

Exercice de td à rajouter

**Procédure :** Nom, arguments, instructions

**Sous-programme :** Nom, instructions, type de retour

**Prototype :** Nom, type de retour, arguments

**La récursivité :** Le modèle de la récursivité consiste en la résolution d’une opération par la résolution de ses composantes. La récursivité est un processus itératif basé sur la référence d’un objet à lui-même.

**Implémentation :** Implémenter de la récursivité en Python implique de définir une fonction qui va s’appeler elle-même.

**Structure :** Une structure de données est un type de variable dont le rôle est de stocker un ensemble d’éléments.

**Itératif et récursif :**

L’itératif est la répétition conditionnée d’un ensemble d’instructions.

Récursif : appel mutuel et successif de fonctions

En récursif, la condition d’arrêt va être basée sur la valeur des arguments d’entrée

Itératif : boucles conditionnées

En itératif, la condition d’arrêt est définie explicitement au début de la boucle

**Test unitaire :**

Vérifier les fonctionnalités de base de chaque fonction du code Vérifier les cas classiques

Anticiper et tester les cas d’erreur

**Test fonctionnel :**

Vérifier le comportement et la validité des fonctions du programme Composition de tests unitaires

**Test de performance :**

Vérifier le temps de réponse/de calcul d’un programme

**Test d’intégration :**

Vérifier la conformité d’un module avec un projet dans son ensemble

**Test de non-régression :**

Vérifier la conformité d’une version ou d’un module avec les versions précédentes

**Tri à bulles :**

Trier un tableau en faisant remonter successivement chaque valeur la plus haute

**Tri rapide (Quicksort) :**

Fonctionnement par pivot : placer tous les éléments inférieurs à gauche et supérieurs à droite

**Tri fusion :**

Tout tableau trié peut être vu comme la fusion de 2 sous-tableaux triés Chaque sous-tableau peut être vu comme la fusion de deux sous-tableaux triés

**Complexité :**

c(n + 1) = c(n) → Complexité constante o(1)

c(n + 1) = c(n) + x → x constante, Complexité linéaire o(n)

c(n + 1) = c(n) + n → Complexité quadratique o(n2)

c(n + 1) = c(n) + 2 ∗ n → Complexité cubique o(n3)

c(n + 1) = c(n) + k ∗ n → Complexité polynomiale

o(nk−1) c(n + 1) = c(n) ∗ 2 → Complexité exponentielle o(2n)

Liste :

lst = [‘a’, ‘b’, ‘c’]

En-tête : lst = [‘new’] + lst

En queue : lst.append(‘new’) // ["a","b","c","new"]

lst.extend(‘new’) // ["a","b","c",["new"]]

Indexé : lst.insert(‘new’, 2) // ["a","b","new","c"]

lst.in(“c”) //True or False

lst.index(“b”) // 1 or 0

Suppression: lst.pop(2) // ["a","b"]

lst.remove(“b”) // ["a","c"]

myStr = "idolovePythonverymuch"

print myStr[0:5] // "idolo"

**Dictionnaire :**

ages={"bob":15,"alice":20, "dave:48","miranda":67"}

"bob" in ages // Renvoie True

"bill" in ages // Returns False

list(ages.keys()) // ["bob","alice","dave","miranda"]

a[1,5,3,2,4]

sorted(a) // [1,2,3,4,5]

a.sort() // Auto assignation

**Chdir :**

from os import chdir

chdir("/home/bob/test")

f = open("file.txt","paramètre")

r → lecture

w → écriture

a → écriture en fin de fichier. Création si fichier inexistant. b → fichier binaire

t → fichier texte

x → création fichier pour écriture.

for line in allLines :

print line //lecture de fichier

f.write("HelloWorld!\n")

f.seek(10) //Changer le curseur de place

f.close()