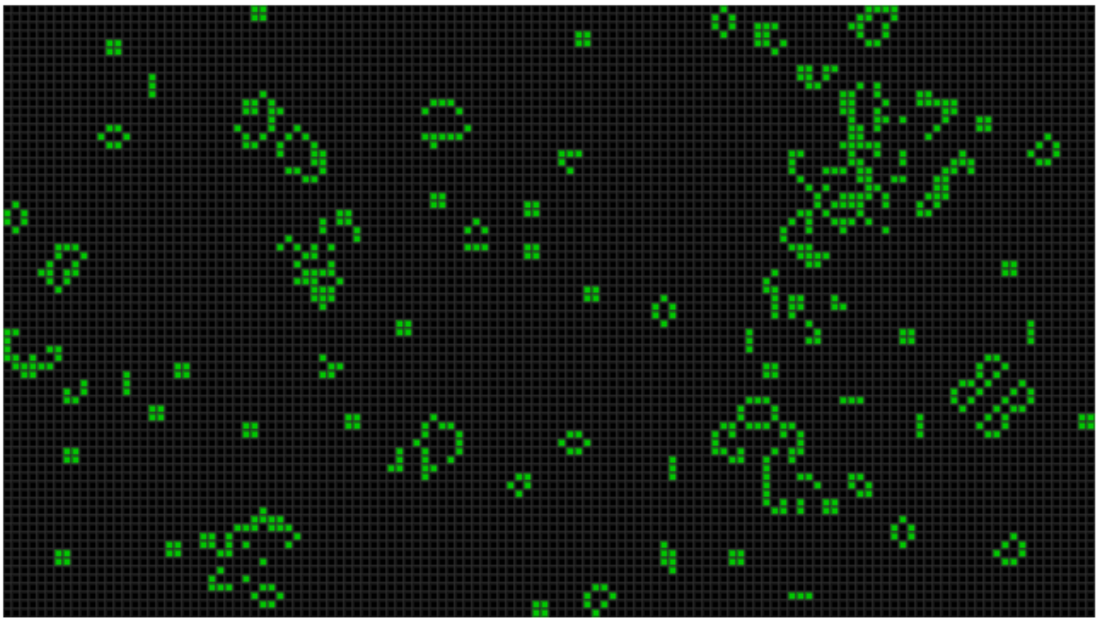
Atelier: Introduction à la programmation créatives avec Processing

Objectif:
Cette formation vise à vous familiariser avec les fondamentaux de la programmation dans l'environnement Processing. Au cours de celle-ci, nous explorerons la structure d'un "sketch", la création de formes simples, la gestion des couleurs, l'exportation de fichiers, les interactions utilisateur, ainsi que les divers types de variables.

La programmation: l'art de communiquer avec les machines.
La programmation est l'art d'écrire des instructions pour un ordinateur afin de résoudre des problèmes ou d'automatiser des tâches. Elle implique la **création de code source**, traduit ensuite en langage machine, permettant à la machine d'exécuter des actions spécifiques.

Processing.org
Créé par Ben Fry et Casey Reas au MIT Media Lab, Processing est un langage de programmation créatif basé sur Java conçu pour simplifier la **création d'œuvres visuelles interactives**. Il offre une syntaxe conviviale et des outils graphiques puissants, permettant aux artistes et développeurs de concevoir rapidement des animations, des visualisations et des installations interactives.

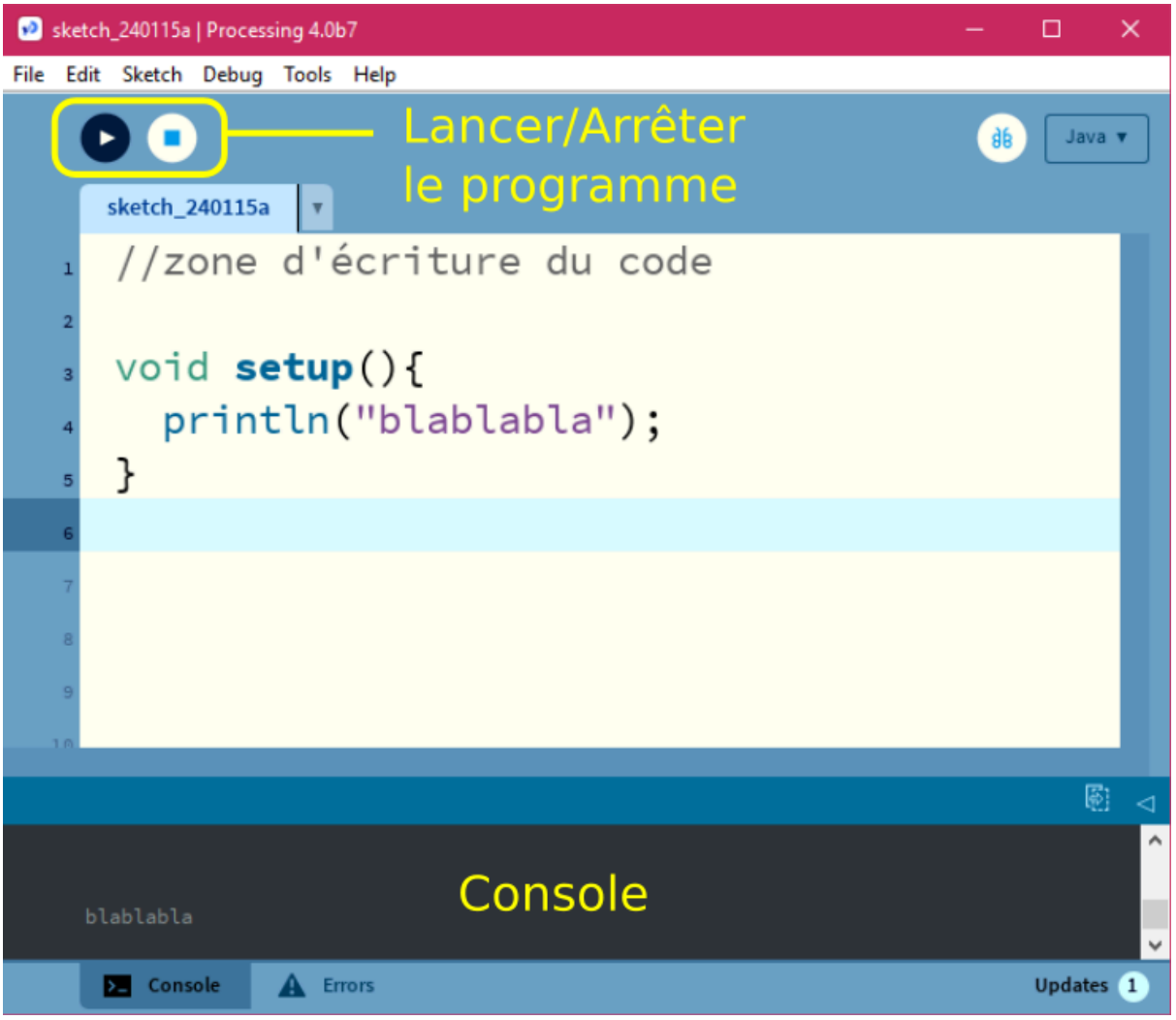


https://processing.org/examples/gameoflife.html

Installation

-Télécharger le programme (processing.org/)
-Installer
-Ouverture de l'application

Interface



Structure d'un sketch processing

Un sketch Processing, qui est un programme écrit dans le langage de programmation Processing, suit une **structure spécifique**. Voici les principales parties d'un sketch Processing :

Fonction "setup()" :
La fonction setup() est appelée **une seule fois au début de l'exécution du programme**. Elle est utilisée pour effectuer les configurations initiales, comme la définition de la taille de la fenêtre, le chargement de ressources (images, vidéos, fichier texte, etc).

```
1  void setup() {
2    size(800, 600); // Définit la taille de la fenêtre
3  }
```

Fonction "draw()" :

La fonction draw() est appelée en continu après la fonction setup(). Elle est utilisée pour effectuer les opérations graphiques et de rendu, créant une animation ou une mise à jour constante de l'affichage. **L'ordre du code est important!**

background(0 - 255) pour niveaux de gris
background(0 - 255, 0-255, 0-255) pour préciser la couleur (rouge, vert, bleu)

ellipse(coordonnéeX, coordonnéeY, largeur, hauteur);

```
5  void draw() {
6    background(255); // Remplit l'arrière-plan en blanc
7    ellipse(50, 50, 30, 30); // Dessine un cercle
8  }
```

Fonctions d'Événement (optionnelles) :

Processing permet également la définition de fonctions pour gérer des événements spécifiques, tels que la souris ou le clavier. Par exemple, les fonctions mousePressed() ou keyPressed() sont appelées lorsque l'utilisateur interagit avec la souris ou le clavier.

```
11 void mousePressed() {
12   // Code à exécuter lorsqu'un clic de souris est détecté
13 }
```

Exécuter le code!

Variables mouseX et mouseY
Dans Processing, mouseX et mouseY sont des variables prédéfinies qui stockent les coordonnées actuelles de la souris dans la fenêtre graphique. Ces variables sont automatiquement mises à jour à chaque itération de la fonction draw() en fonction de la position de la souris.

mouseX : Contient la position horizontale (coordonnée x) de la souris.
mouseY : Contient la position verticale (coordonnée y) de la souris.

Exercice:
-Remplacer les coordonnées X et Y dans “ellipse” par mouseX et mouseY respectivement.
-Transférer la fonction “background(255);” vers la fonction événement “mousePressed()”.

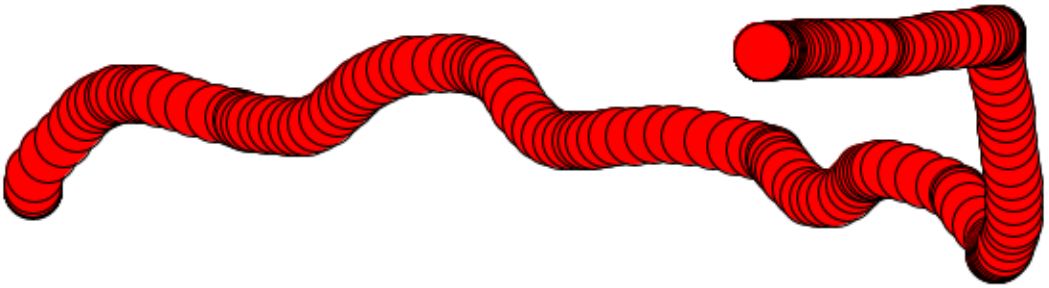
Exemple:

```
void setup() {
  size(800, 600); // Définit la taille de la fenêtre
}

void draw() {

  ellipse(mouseX, mouseY, 30, 30); // Dessine un cercle
}


void mousePressed() { //Code à executer lorsqu'un clic de souris est détecté
    background(255); // Remplit l'arrière-plan en blanc
}
```

Exécuter le code!

Pour ajouter de la couleur, insérez la fonction "fill(Rouge, Vert, Bleu);" avant la ligne où le cercle est dessiné. Par exemple :

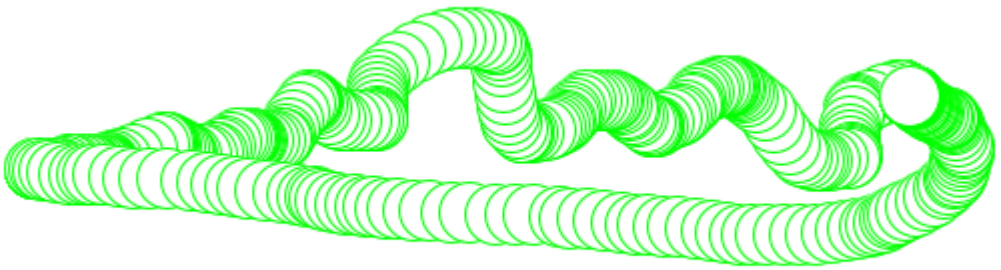
```
9   fill(255,0,0); // pour un cercle rouge
10  ellipse(mouseX, mouseY, 30, 30); // Dessine un cercle
```



Vous pouvez personnaliser la couleur en ajustant les valeurs Rouge, Vert et Bleu (0-255) selon vos préférences.

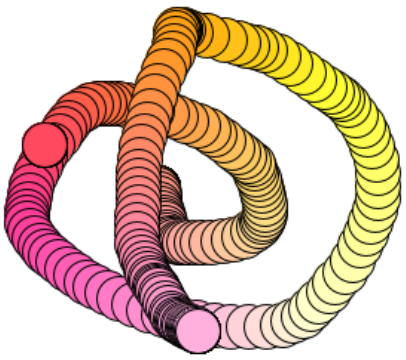
Pour changer la couleur du contour, utilisez la fonction "stroke(Rouge, Vert, Bleu);"

```
10  stroke(0,255,0); // pour un contour vert
11  ellipse(mouseX, mouseY, 30, 30); // Dessine un cercle
```



Insertion de la position de la souris (mouseX, mouseY) dans la fonction “fill()”.

```
8  void draw() {
9    fill(255,mouseX,mouseY); // pour un cercle rouge
10   ellipse(mouseX, mouseY, 30, 30); // Dessine un cercle
```



Exportation d'image
Sauvegarder l'image affichée en ajoutant la fonction keyPressed(){ } et en y ajoutant saveFrame("nom.jpg");.

```
void keyPressed() {
  // La fonction saveFrame() enregistre un instantané de la fenêtre de dessin.
  // Dans ce cas, elle enregistre l'instantané en tant que fichier image avec le nom "image.jpg".
  saveFrame("image.jpg");
}
```

Période d'exploration (fonctions supplémentaires #1)

Plusieurs fonctions sont couramment utilisées pour effectuer des opérations graphiques, interactives et de traitement de données. Voici quelques-unes des fonctions les plus fréquemment utilisées :

Graphiques de base :
- **line(x1, y1, x2, y2)**: Trace une ligne entre deux points.
- **ellipse(x, y, largeur, hauteur)**: Dessine une ellipse ou un cercle.
- **rect(x, y, largeur, hauteur)**: Dessine un rectangle.
- **background(couleur)**: Définit la couleur de fond de la fenêtre.
- **fill(couleur)**: Définit la couleur de remplissage pour les formes suivantes.
- **stroke(couleur)**: Définit la couleur de la bordure des formes suivantes.

Interaction utilisateur :
- **mouseX et mouseY**: Variables contenant les coordonnées actuelles de la souris.
- **mousePressed()**: Fonction appelée lorsqu'un clic de souris est détecté.
- **keyPressed()**: Fonction appelée lorsqu'une touche du clavier est enfoncée.

Temps et Animation :
- **frameCount**: Variable contenant le nombre d'images affichées depuis le début de l'exécution.
- **frameRate(fps)**: Définit le taux de rafraîchissement de l'animation.

Mathématiques :
- **random(min, max)**: Génère un nombre aléatoire entre min (inclus) et max (exclus).
- **map(value, start1, stop1, start2, stop2)**: Réattribue une valeur d'une plage à une autre.
- **sin(angle), cos(angle), tan(angle)**: Fonctions trigonométriques.

Texte :
- **text(txt, x, y)**: Affiche du texte à la position spécifiée.
- **textSize(size)**: Définit la taille du texte.
- **textAlign(alignment)**: Définit l'alignement du texte.

Ces fonctions offrent des fonctionnalités de base pour la création d'animations, d'interactions utilisateur et de visualisations graphiques dans un programme Processing. La documentation officielle de Processing est une ressource précieuse pour explorer davantage ces fonctions et en découvrir de nouvelles.

Variables

En Processing, une variable est un espace de stockage associé à un nom, permettant de contenir des données comme des nombres, des caractères ou des chaînes de caractères. Les variables sont utilisées pour stocker et manipuler l'information tout au long de l'exécution d'un programme.

```
int x = 100;
float y = 3.14;
String nom = "Processing";
```

Pour **déclarer** une variable, il vous suffit de spécifier son type (int, float, String, etc.), suivi de son nom, puis du symbole "=" et enfin de la valeur souhaitée.

Variable Globale :

Une variable globale est déclarée en dehors de toutes les fonctions dans un programme, ce qui signifie qu'elle est accessible depuis n'importe quelle partie du code, y compris à l'intérieur des fonctions. Elle conserve sa valeur même après la sortie de la fonction où elle a été définie, et son accès peut être étendu à l'ensemble du programme.

```

int variableGlobale = 10; // Variable globale

void setup() {
  //  Utilisation de la variable globale ici
}

void draw() {
  // et ici
}
```

Variable Locale :

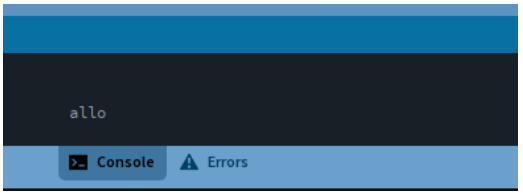
Une variable locale est déclarée à l'intérieur d'une fonction, et elle est accessible uniquement à l'intérieur de cette fonction. Elle n'est pas visible en dehors de la fonction où elle a été définie.La portée d'une variable locale est limitée au bloc de code (la fonction) dans lequel elle a été créée.

```
void setup() {
  int variableLocale = 5; // Variable locale à la fonction setup
  // ...
}

void draw() {
  // La variableLocale n'est pas accessible ici
}
```

Console

La console dans l'IDE (Environnement de Développement Intégré) de Processing est une fenêtre qui affiche des messages générés par le programme pendant son exécution. Elle sert à des fins de débogage, de suivi des étapes d'exécution, et à l'affichage d'informations utiles. On utilise la fonction “println()” pour afficher des messages dans la console.

```
void mousePressed(){
  println("allo");
}
```



Boucle “for()”

Une boucle "for" est une structure de contrôle de flux utilisée en programmation pour répéter l'exécution d'un bloc de code un nombre spécifié de fois. La boucle "for" est particulièrement utile lorsque le nombre d'itérations est connu à l'avance.

La syntaxe générale d'une boucle "for" en langage de programmation Processing (et dans de nombreux autres langages) est la suivante :

```
for (initialisation ; condition ; itération) {
  // Bloc de code à répéter
}
```

Initialisation : Cette partie est exécutée une seule fois au début de la boucle. Elle est souvent utilisée pour initialiser une variable de comptage.

Condition : La condition est évaluée avant chaque itération. Si la condition est vraie, le bloc de code à l'intérieur de la boucle est exécuté. Si la condition est fausse, la boucle se termine.

Itération : Cette partie est exécutée à la fin de chaque itération de la boucle. Elle est généralement utilisée pour mettre à jour la variable de comptage.

```
void setup() {
  size(400, 200);
}

void draw() {
  background(255);

  // Boucle for pour dessiner 10 cercles espacés horizontalement
  for (int i = 0; i < 10; i++) {
    ellipse(40 + i * 40, height/2, 30, 30);
  }
}
```

Dans cet exemple, la boucle "for" est utilisée pour dessiner 10 cercles horizontalement. La variable i est utilisée comme compteur, elle est initialisée à 0, la condition est que i doit être inférieur à 10, et i est incrémentée à chaque itération. Le résultat est que le bloc de code à l'intérieur de la boucle est exécuté 10 fois, dessinant ainsi 10 cercles espacés régulièrement. **Notez la formulation!**

Conditions pour une boucle “for”

for(int i = 0; i < 10; i++){code}

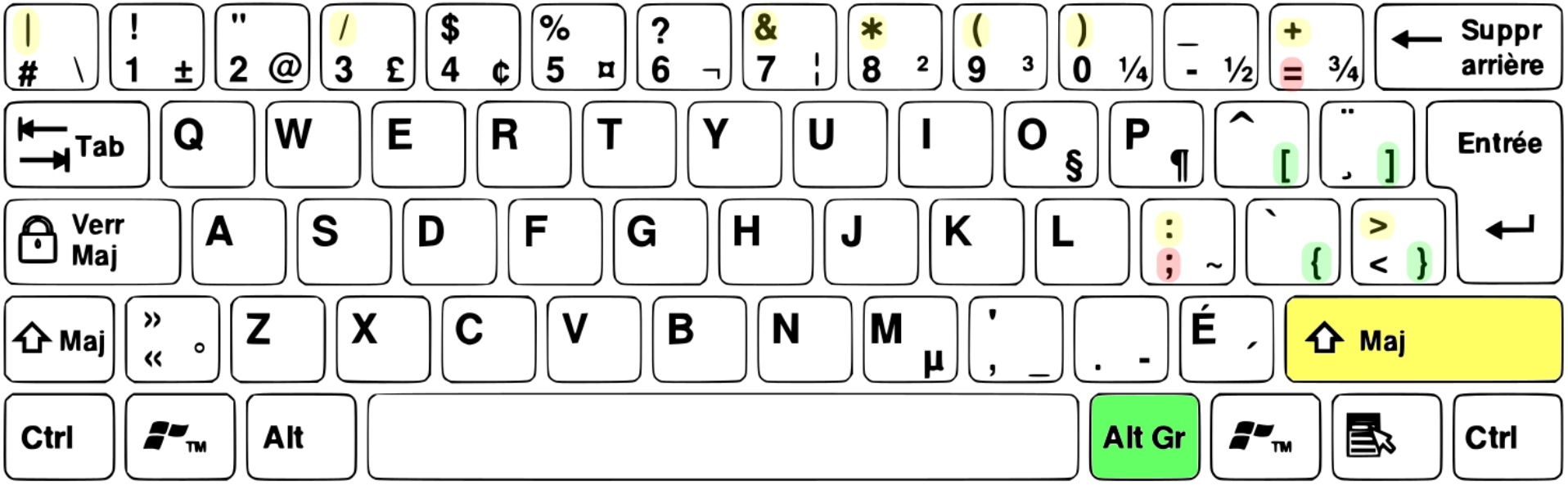
Conditions
== (égal à) : Vérifie si deux valeurs sont égales.
!= (différent de) : Vérifie si deux valeurs ne sont pas égales.
< (inférieur à) : Vérifie si une valeur est inférieure à une autre.
<= (inférieur ou égal à) : Vérifie si une valeur est inférieure ou égale à une autre.
> (supérieur à) : Vérifie si une valeur est supérieure à une autre.
>= (supérieur ou égal à) : Vérifie si une valeur est supérieure ou égale à une autre.

Itération
++ (Incrémentation de 1) :
Exemple : i++ ajoute 1 à la valeur de la variable i.
Utilisation : i++ est équivalent à i += 1 ou i = i + 1.

+= (Incrémentation par une valeur) :
Exemple : i += 2 ajoute 2 à la valeur de la variable i.
Utilisation : i += n ajoute la valeur de n à la variable i.

-- (Décrémentation de 1) :
Exemple : i-- soustrait 1 de la valeur de la variable i.
Utilisation : i-- est équivalent à i -= 1 ou i = i - 1.

-= (Décrémentation par une valeur) :
Exemple : i -= 3 soustrait 3 de la valeur de la variable i.
Utilisation : i -= n soustrait la valeur de n de la variable i.


| # \
! 1 ±
" 2 @
/ 3 £
$ 4 ¢
% 5 ¤
? 6 ¬
& 7 ¦
* 8 ²
(9 ³
) 0 ¼
_ - ½
+ = ¾
Suppr arrière
Tab
Q
W
E
R
T
Y
U
I
O §
P ¶
^ [
¨] ¸
Entrée
Verr Maj
A
S
D
F
G
H
J
K
L
: ; ~
` {
> < }
Maj
» « º
Z
X
C
V
B
N
M µ
' , _
. -
É ´
Maj
Ctrl
Alt
Alt Gr
Ctrl