IT技術特待生　公開講座

コードで創る未来
顔認識×猫耳
プログラミング体験

あなたのプログラミング能力が
IT技術特待生への扉を開く

```
> init_system()
> loading_modules...
> start_workshop(mode="creative")
```

今日の体験で得られるもの

プログラミングで「創る楽しさ」を体感しよう

即座のフィードバック

書いたコードがすぐに画面上で動き、視覚的な結果として現れる喜びを体験。
エラーも成功も、その場ですぐに分かります。

段階的な成長

基礎から応用へ、ステップバイステップで機能を追加していく達成感。
小さな成功体験の積み重ねが自信になります。

実践的スキル

カメラ制御、画像認識、グラフィック処理。
現代のIT業界で求められる技術の基礎を、手を動かして学びます。

アイデアを積み重ねて、より良いものを創りあげよう！

今日のゴール
猫耳カメラアプリを作る

90分で完成させる
「顔認識×猫耳AR」プログラム

Step 1: カメラ映像の取得
PC搭載のカメラから映像を取り込み表示する

Step 2: 顔認識機能の導入
AIモデルを使って顔の位置を検出する

Step 3: 猫耳画像の合成
検出した顔に合わせて猫耳を配置する

Step 4: もくもくタイム
演出改善や機能追加など、あなただけのアプリとして作りこむ

Step 1:
カメラ映像の取得

PC搭載のカメラから映像を取り込み表示する

```
> init_system()
> loading_modules...
> start_workshop(mode="creative")
```

Step 1
カメラプログラムの基礎

まずはカメラ映像を取得してみよう

Webカメラから映像を取得し、画面に表示してみましょう。
PythonとOpenCVライブラリを使用します。

- ライブラリのインポート
- ループ処理によるリアルタイム取得
- ボタン押下による制御

camera.py（一部抜粋） Python

```python
# カメラの初期化
cap = cv2.VideoCapture(CAMERA_INDEX)
if not cap.isOpened():
    print(f"エラー: カメラ（インデックス {CAMERA_INDEX}）を開けませんでした。")
    print("カメラが接続されているか、他のアプリケーションで使用されていないか確認してください。")
    root.destroy()
    exit()

# 映像表示用のパネル（Labelウィジェット）
panel = tk.Label(root)
panel.pack(padx=10, pady=10)

# 撮影ボタン
capture_button = tk.Button(root, text="撮影", command=capture_image)
capture_button.pack(side=tk.LEFT, padx=5, pady=5)

# ライブフィード再開ボタン（初期状態は無効）
resume_button = tk.Button(root, text="ライブフィード再開", command=resume_live_feed, state=tk.DISABLED)
resume_button.pack(side=tk.RIGHT, padx=5, pady=5)

# ウィンドウが閉じられたときのプロトコルを設定
root.protocol("WM_DELETE_WINDOW", on_closing)

# 最初のフレーム更新を開始
update_frame()

# Tkinterイベントループを開始
root.mainloop()
```

今回は事前に用意したプログラムを元に動作を確認していきます

Step 1 – カメラの動作確認

正しく表示されるか動作確認してみよう

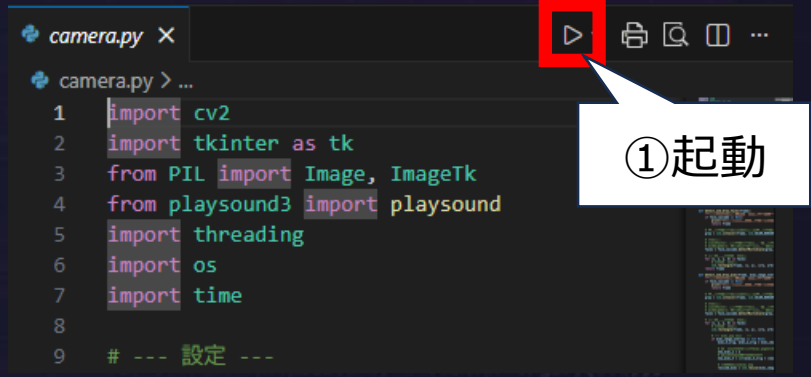
確認の手順

- **①プログラムの起動（図１）**
  VSCode上の実行ボタンをクリック。
  カメラプログラムが起動します。

- **②表示されている映像を撮影（図２）**
  カメラプログラム上の「撮影」ボタンをクリック。
  画面が停止し、撮影した映像が画像ファイルとして保存されます。

- **③撮影モードに復帰（図２）**
  「ライブフィード再開」ボタンをクリック。
  Webカメラに写っている映像の表示が再開することを確認します。

- **④撮影した画像ファイルの確認（図３）**
  VSCode左部の画像ファイル名をクリック。
  撮影した映像が表示されることを確認します。



プログラムの起動（図１）

Step 1 – カメラの動作確認

正しく表示されるか動作確認してみよう

確認の手順

- **①プログラムの起動（図１）**
  VSCode上の実行ボタンをクリック。
  カメラプログラムが起動します。

- **②表示されている映像を撮影（図２）**
  カメラプログラム上の「撮影」ボタンをクリック。
  画面が停止し、撮影した映像が画像ファイルとして保存されます。

- **③撮影モードに復帰（図２）**
  「ライブフィード再開」ボタンをクリック。
  Webカメラに写っている映像の表示が再開することを確認します。

- **④撮影した画像ファイルの確認（図３）**
  VSCode左部の画像ファイル名をクリック。
  撮影した映像が表示されることを確認します。

Webカメラでの撮影（図２）

Step 1 – カメラの動作確認

正しく表示されるか動作確認してみよう

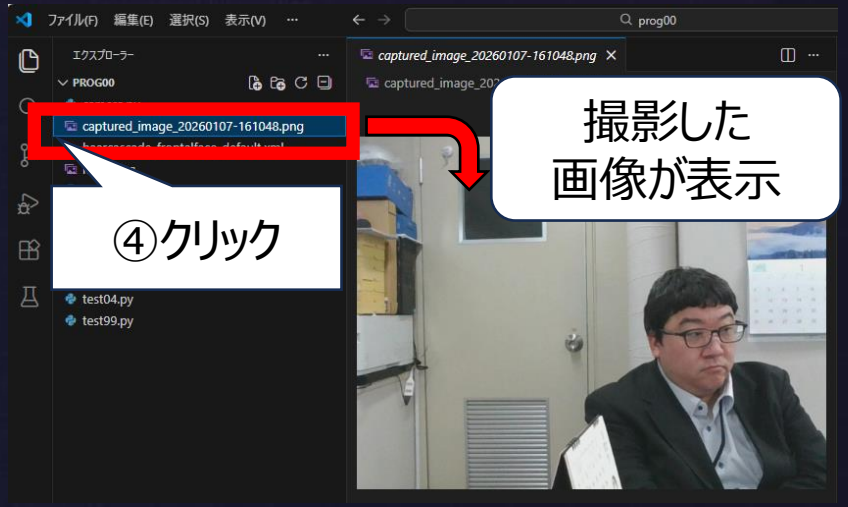
確認の手順

- **①プログラムの起動（図１）**
  VSCode上の実行ボタンをクリック。
  カメラプログラムが起動します。

- **②表示されている映像を撮影（図２）**
  カメラプログラム上の「撮影」ボタンをクリック。
  画面が停止し、撮影した映像が画像ファイルとして保存されます。

- **③撮影モードに復帰（図２）**
  「ライブフィード再開」ボタンをクリック。
  Webカメラに写っている映像の表示が再開することを確認します。

- **④撮影した画像ファイルの確認（図３）**
  VSCode左部の画像ファイル名をクリック。
  撮影した映像が表示されることを確認します。



撮影した画像ファイルの確認（図３）

Step 2:
顔認識機能の導入

AIモデルを使って顔の位置を検出する

```
> init_system()
> loading_modules...
> start_workshop(mode="creative")
```

Step 2 - 顔認識機能の仕組み

AIの力で顔を検出する

OpenCVには事前学習済みの顔認識モデル(Haar Cascade)が含まれています。
これを使うことで、複雑な計算なしに顔の位置を特定できます。

camera.py（一部抜粋） Python

```python
# グレースケールに変換（顔検出はグレースケールで行うのが一般的）
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# 顔を検出
# scaleFactor: 画像スケールをどれだけ縮小するか。1.1は10%縮小。
# minNeighbors: 候補となる矩形がどれだけの近傍を持つべきか。小さい値は誤検出を増やすが、検出率も上がる。
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# 検出された顔に赤枠を描画
for (x, y, w, h) in faces:
    # 顔に赤枠を描画
    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2) # (0, 0, 255)はBGRで赤色、2は線の太さ
```

機械学習モデル
事前学習済みデータの活用

座標とサイズ
(x, y)座標と幅・高さの取得

可視化
検出結果を矩形で描画

Step 2 – 顔認証機能の実装

ソースコードを修正して顔認証機能を追加してみよう

追加手順

- **①顔検出器の初期化（図４）**
  プログラム起動時に顔検出器を作成する様にプログラムを追加します。
- **②撮影時処理を追加（図５）**
  カメラプログラム上の「撮影」ボタンをクリックした際に
  撮影した画像から顔の部位を検出し、赤枠で囲む処理を追加します。
- **③起動時の動作確認（図６）**
  VSCode上の実行ボタンをクリック。
  プログラムが起動し、追加前と同様にカメラに映った映像が表示されます。
- **④撮影時処理の動作確認（図７）**
  撮影ボタンをクリックし、写っている画像が撮影されることを確認。
  映像の顔の部分が赤枠で囲われていることを確認します。

220行目に以下を追加。
face_cascade =
cv2.CascadeClassifier(HAARCASCADE_PATH)

```
215   # --- メインアプリケーションのセットア
216   root = tk.Tk()
217   root.title("顔認識×猫耳カメラ")
218
219   # 顔検出器の初期化
220   face_cascade = cv2.CascadeClassifier(HAARCASCADE_PATH)
```

顔検出器の初期化（図４）

Step 2 – 顔認証機能の実装

ソースコードを修正して顔認証機能を追加してみよう

追加手順

- **①顔検出器の初期化（図４）**
  プログラム起動時に顔検出器を作成する様にプログラムを追加します。

- **②撮影時処理を追加（図５）**
  カメラプログラム上の「撮影」ボタンをクリックした際に
  撮影した画像から顔の部位を検出し、赤枠で囲む処理を追加します。

- **③起動時の動作確認（図６）**
  VSCode上の実行ボタンをクリック。
  プログラムが起動し、追加前と同様にカメラに映った映像が表示されます。

- **④撮影時処理の動作確認（図７）**
  撮影ボタンをクリックし、写っている画像が撮影されることを確認。
  映像の顔の部分が赤枠で囲われていることを確認します。

176～177行目に以下を追加。
face_frame = current_frame.copy()
face_frame = detect_and_draw_faces(face_frame)

```
# 撮影したフレームのコピーを作成し、顔
face_frame = current_frame.copy()
face_frame = detect_and_draw_faces(face_frame)

# 画像をファイルに保存（タイムスタンプを付けてユニークなファイル名にする）
timestamp = time.strftime("%Y%m%d-%H%M%S")
filename = f"captured_image_{timestamp}.png"
cv2.imwrite(filename, face_frame)
print(f"画像を保存しました:

# 撮影した画像をTkinterで表示
captured_cv2image = cv2.cvtColor(face_frame, cv2.COLOR_BGR2RGB)
captured_pil_image = Image.fromarray(captured_cv2image)
captured_imgtk = ImageTk.Phot
```

current_frameからface_frameに変更

current_frameからface_frameに変更

撮影時処理を追加（図５）

Step 2 – 顔認証機能の実装

ソースコードを修正して顔認証機能を追加してみよう

追加手順

- **①顔検出器の初期化（図４）**
  プログラム起動時に顔検出器を作成する様にプログラムを追加します。

- **②撮影時処理を追加（図５）**
  カメラプログラム上の「撮影」ボタンをクリックした際に
  撮影した画像から顔の部位を検出し、赤枠で囲む処理を追加します。

- **③起動時の動作確認（図６）**
  VSCode上の実行ボタンをクリック。
  プログラムが起動し、追加前と同様にカメラに映った映像が表示されます。

- **④撮影時処理の動作確認（図７）**
  撮影ボタンをクリックし、写っている画像が撮影されることを確認。
  映像の顔の部分が赤枠で囲われていることを確認します。

起動時の動作確認（図６）

Step 2 – 顔認証機能の実装

ソースコードを修正して顔認証機能を追加してみよう

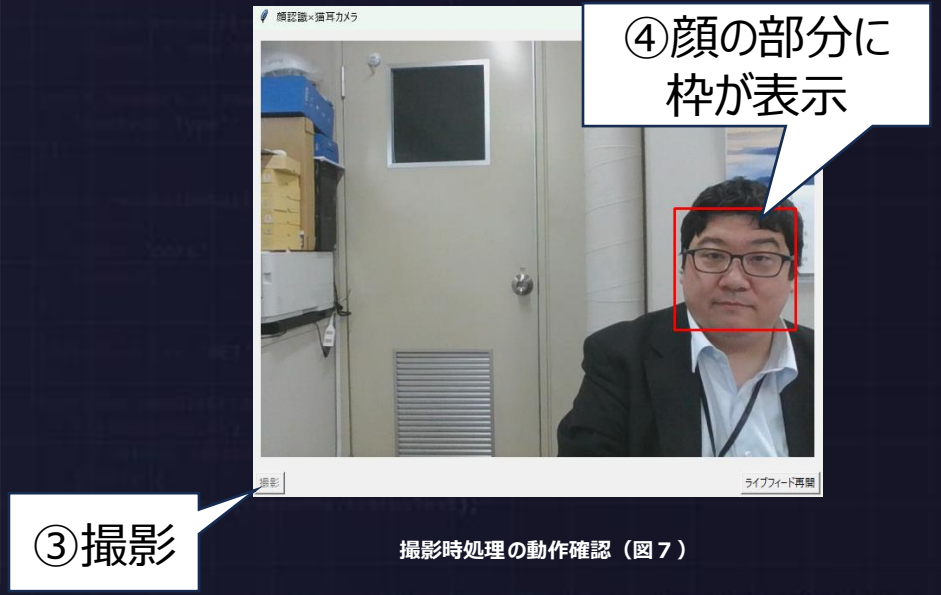
追加手順

- **①顔検出器の初期化（図４）**
  プログラム起動時に顔検出器を作成する様にプログラムを追加します。

- **②撮影時処理を追加（図５）**
  カメラプログラム上の「撮影」ボタンをクリックした際に
  撮影した画像から顔の部位を検出し、赤枠で囲む処理を追加します。

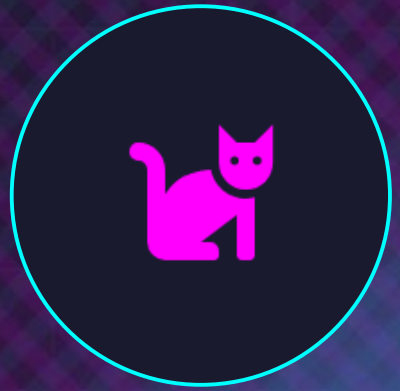
- **③起動時の動作確認（図６）**
  VSCode上の実行ボタンをクリック。
  プログラムが起動し、追加前と同様にカメラに映った映像が表示されます。

- **④撮影時処理の動作確認（図７）**
  撮影ボタンをクリックし、写っている画像が撮影されることを確認。
  映像の顔の部分が赤枠で囲われていることを確認します。



撮影時処理の動作確認（図７）

Step 3:
猫耳画像の合成



検出した顔に合わせて猫耳を配置する

```
> init_system()
> loading_modules...
> start_workshop(mode="creative")
```

Step 3 - 猫耳の合成

検出した顔に猫耳を乗せてみよう

実装のポイント

猫耳画像の準備
背景が透明なPNG画像を使用します。

位置計算 (Coordinate Calculation)
顔の座標(x, y)と幅(w)を基準に、猫耳の中心を合わせます。

画像合成 (Alpha Blending)
元の映像に猫耳画像を重ね合わせます。

camera.py（一部抜粋） Python

```python
# 赤枠の横幅に合わせてmimi.pngをリサイズ
new_mimi_w = w
# 縦横比を維持して高さを計算
new_mimi_h = int(mimi_h_orig * (new_mimi_w / mimi_w_orig))
```

SYSTEM_READY

Step 3 – 猫耳画像の合成

ソースコードを修正して猫耳画像を合成して表示してみよう

追加手順

- **①猫耳画像の読み込み（図８）**
  撮影した写真に合成する猫耳画像を読み込む処理を追加します。
- **②撮影時処理の更新（図９）**
  「撮影」ボタンをクリックした際に合成処理を行える様プログラムを変更します。
- **③起動時の動作確認（図10）**
  VSCode上の実行ボタンをクリック。
  プログラムが起動し、追加前と同様にカメラに映った映像が表示されます。
- **④撮影時処理の動作確認（図11）**
  撮影ボタンをクリックし、写っている画像が撮影されることを確認。
  映像の頭の部分に猫耳が合成されていることを確認します。

```
# 顔検出器の初期化
face_cascade = cv2.CascadeClassifier(HAARCASCADE_PATH)

# mimi.pngの読み込み
mimi_image_global = cv2.imread(MIMI_IMAGE_PATH, cv2.IMREAD_UNCHANGED)
```

猫耳画像の読み込み（図８）

223行目に以下を追加。
mimi_image_global = cv2.imread(MIMI_IMAGE_PATH, cv2.IMREAD_UNCHANGED)

Step 3 – 猫耳画像の合成

ソースコードを修正して猫耳画像を合成して表示してみよう

追加手順

- **①猫耳画像の読み込み（図８）**
  撮影した写真に合成する猫耳画像を読み込む処理を追加します。
- **②撮影時処理の更新（図９）**
  「撮影」ボタンをクリックした際に合成処理を行える様プログラムを変更します。
- **③起動時の動作確認（図10）**
  VSCode上の実行ボタンをクリック。
  プログラムが起動し、追加前と同様にカメラに映った映像が表示されます。
- **④撮影時処理の動作確認（図11）**
  撮影ボタンをクリックし、写っている画像が撮影されることを確認。
  映像の頭の部分に猫耳が合成されていることを確認します。

```
if ret:
    current_frame = frame.copy()  最新のフレームを保存
```

143行目に.copy()を追加

```
# 撮影したフレームのコピーを作成し、顔検出と描画を行う
face_frame = current_frame.copy()
face_frame = detect_and_draw_mimi(face_frame, mimi_image_global)
```

177行目を変更。
draw_facesからdraw_mimiへ

,mimi_image_globalを追加

撮影時処理の更新（図９）

Step 3 – 猫耳画像の合成

ソースコードを修正して猫耳画像を合成して表示してみよう

追加手順

- **①猫耳画像の読み込み（図８）**
  撮影した写真に合成する猫耳画像を読み込む処理を追加します。

- **②撮影時処理の更新（図９）**
  「撮影」ボタンをクリックした際に合成処理を行える様プログラムを変更します。

- **③起動時の動作確認（図10）**
  VSCode上の実行ボタンをクリック。
  プログラムが起動し、追加前と同様にカメラに映った映像が表示されます。

- **④撮影時処理の動作確認（図11）**
  撮影ボタンをクリックし、写っている画像が撮影されることを確認。
  映像の頭の部分に猫耳が合成されていることを確認します。

起動時の動作確認（図10）

Step 3 – 猫耳画像の合成

ソースコードを修正して猫耳画像を合成して表示してみよう

追加手順

- **①猫耳画像の読み込み（図８）**
  撮影した写真に合成する猫耳画像を読み込む処理を追加します。

- **②撮影時処理の更新（図９）**
  「撮影」ボタンをクリックした際に合成処理を行える様プログラムを変更します。

- **③起動時の動作確認（図10）**
  VSCode上の実行ボタンをクリック。
  プログラムが起動し、追加前と同様にカメラに映った映像が表示されます。

- **④撮影時処理の動作確認（図11）**
  撮影ボタンをクリックし、写っている画像が撮影されることを確認。
  映像の頭の部分に猫耳が合成されていることを確認します。

撮影時処理の動作確認（図11）

気になること


顔認識×猫耳カメラ
撮影
ちょっと顔の周りの
枠が邪魔…

ということで

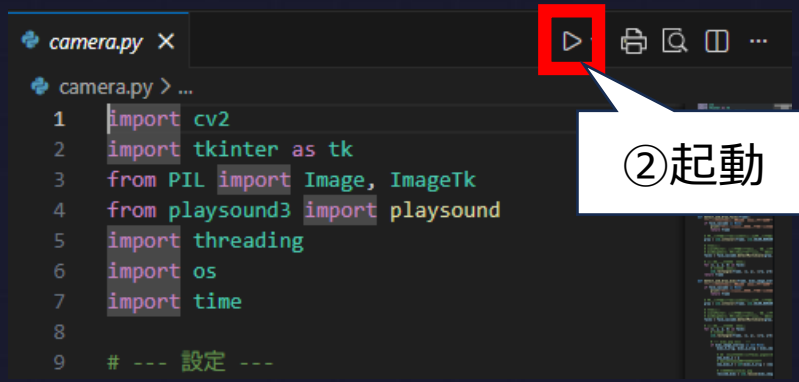
赤枠消しましょう！

Step 3 – 最後の仕上げ

ソースコードを修正して猫耳画像を合成して表示してみよう

```
# 検出された顔に赤枠を描画
for (x, y, w, h) in faces:
    # 顔に赤枠を描画
    #cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2) # (0, 0, 255)はBGRで赤色、2は線の太さ
```

①73行目のcv2.の前に「＃」を追加
（赤枠処理をコメントアウトして動作しない様に変更）



②起動

③撮影を
押すと…？

無事完成！
おめでとうございます！

Step 4:
もくもくタイム

演出改善や機能追加など、あなただけのアプリとして作りこむ

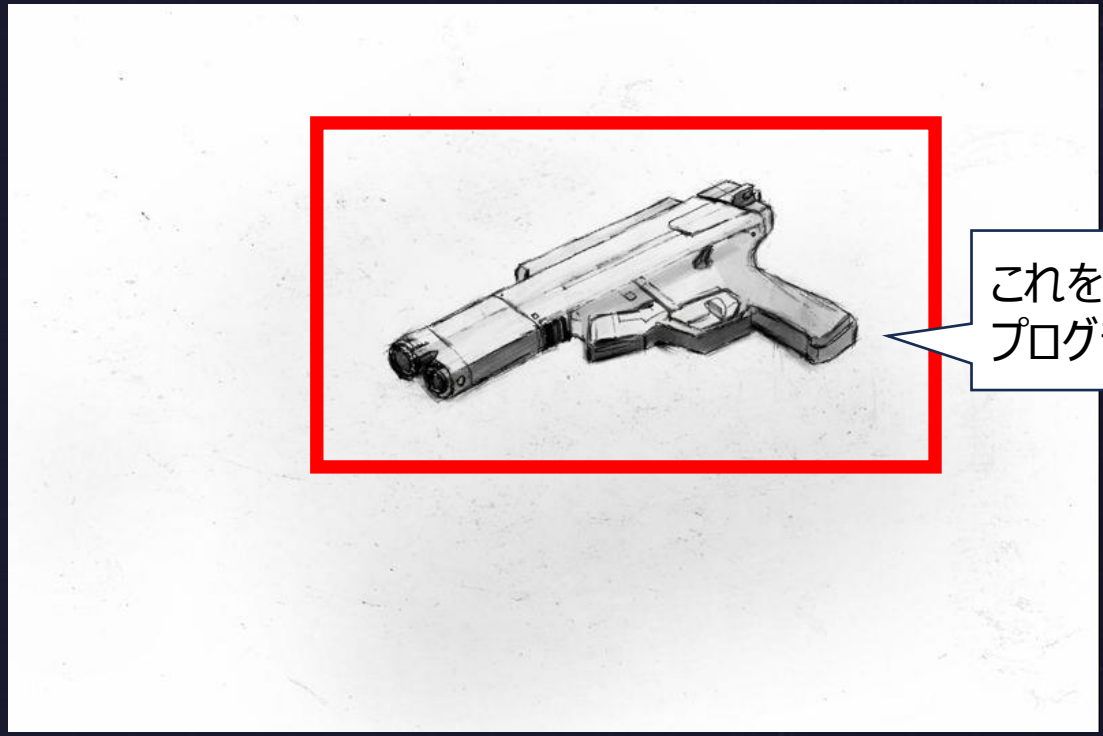
```
> init_system()
> loading_modules...
> start_workshop(mode="creative")
```

Step 4: もくもくタイムとは

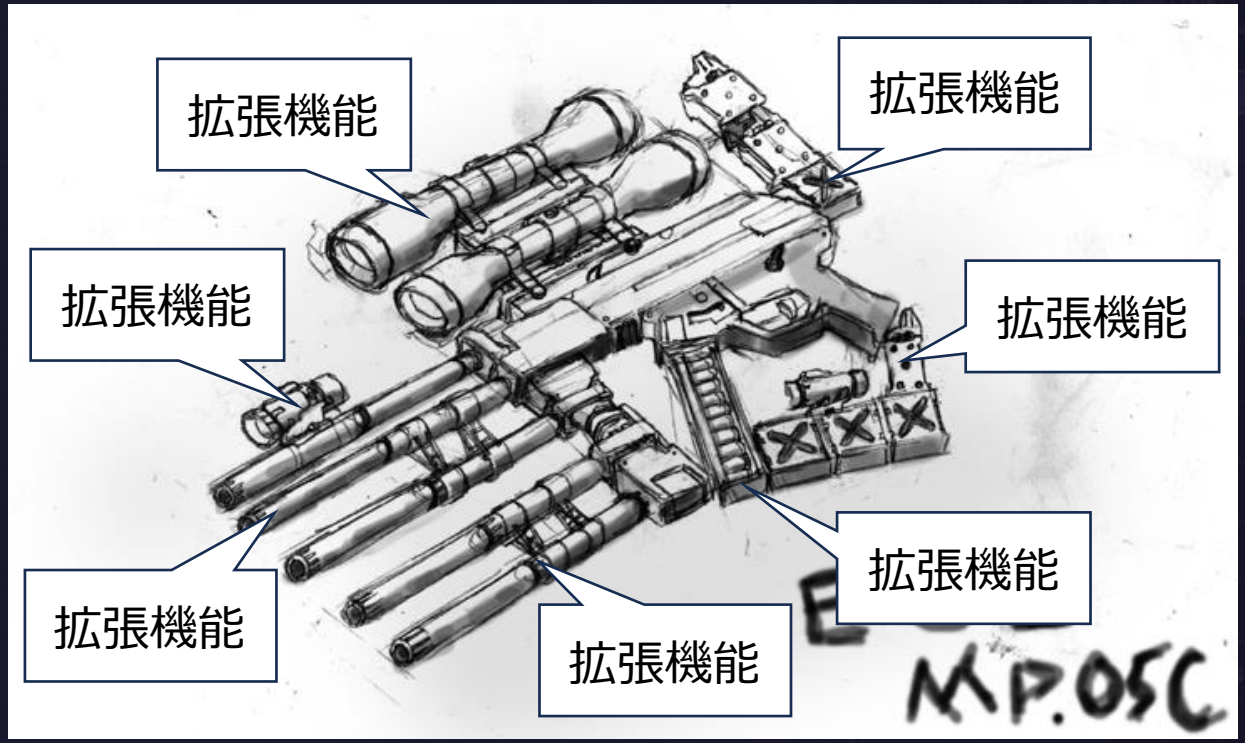
もくもくタイムってそもそも何？

- ITエンジニアの勉強会が語源（集まって黙々と勉強する会＝もくもく会）
- 家ではやることが多く、まとまった時間を取ることが難しい人たちが集中して取り組むための集まり

- 今回は「作ってみたプログラムを もくもくと改造してみよう！」というテーマで取り組みます！



例えば元のプログラムが
あったとした時に



引用：https://wikiwiki.jp/eoe/カスタマイズ例

これくらい魔改造できたら
面白いですよね

Step 4: 更なるカスタマイズに向けたヒント

あなただけのオリジナル機能を追加しよう

LEVEL UP!

エフェクト追加

キラキラパーティクルや、画面全体の色調を変えるフィルター効果を実装。

複数の装飾

猫耳だけでなく、ヒゲや鼻、サングラスなどのアイテムを追加して重ね合わせる。

色変更機能

キーボード入力（'R', 'G', 'B'など）で、猫耳の色を変更。

リアルタイム装飾機能

撮影をしなくてもリアルタイムに装飾をつけてくれる！

> プログラミングの真の楽しさは、**自分のアイデアを形にする**ことです。

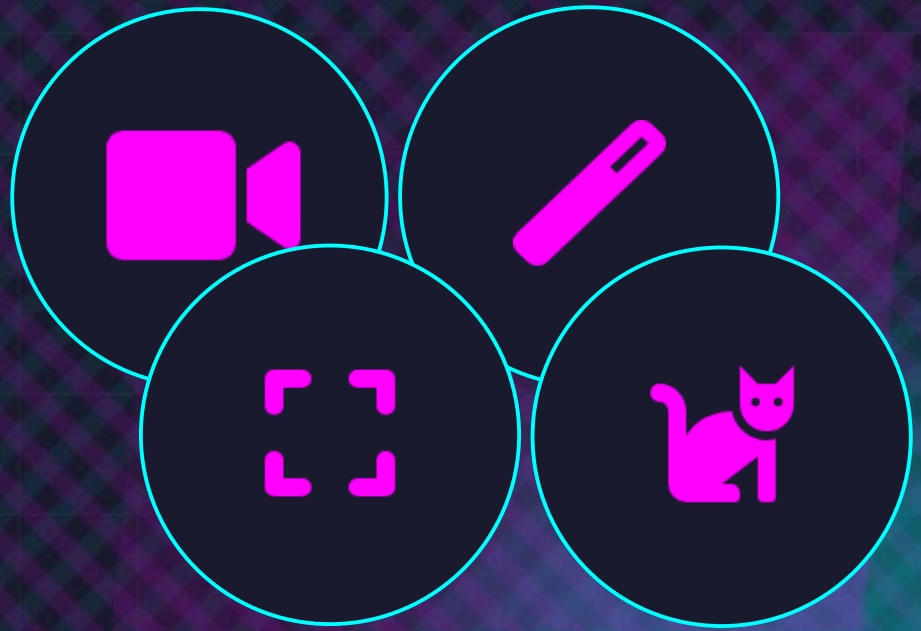
もくもくタイム
～14:20まで

もくもくタイム終了！

お疲れさまでした！

さいごに

今日のまとめやIT特待生制度について



```
> init_system()
> loading_modules...
> start_workshop(mode="creative")
```

プログラミング能力が拓く可能性

今日の体験が示すあなたの未来

顔認識技術は、セキュリティ、AR/VR、医療など、世界中で活用されています。
今日学んだ基礎は、無限の可能性への入り口です。

問題解決力

複雑な課題を小さなステップに分解し、論理的に解決する力。
あらゆる仕事で求められる核心的なスキルです。

創造力

「あったらいいな」を自分の手で形にする実装力。
アイデアを現実のプロダクトに変える魔法です。

市場価値

IT人材不足が続く現代において、高い需要と報酬。
技術力は、あなたのキャリアを強力に支えます。

自己表現

コードを通じて自分の世界観を表現する手段。
アート、ゲーム、ツールなど、表現の幅は無限大です。

このような能力を持つ高校生を、本校はIT技術特待生として歓迎します

特待生選考制度について

あなたのプログラミング能力を評価します

特待生特典 & 選考特徴

実技重視の評価

ペーパーテストではなく、実際にコードを書く能力を重視。

段階的評価システム

基礎から応用まで、あなたのレベルに応じた課題で評価。

最大146万円の授業料減免

優秀者には経済的な支援と特別プログラムへの参加資格を提供。

評価基準

EVALUATION_METRICS

- コードの正確性と効率性
- 問題解決へのアプローチ

コードで創る
あなたの未来

特待生選考でお待ちしています

学校法人岩崎学園　情報科学専門学校