

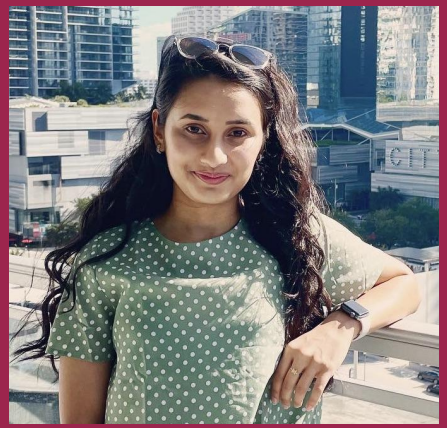



Reptile

Aileen Cano, Aviva Weinbaum, Lindsey Weiskopf, Hariti Patel

Meet the Team

Aileen Cano
Test Designer
Identifies with Turtles



Hariti Patel
Team Manager
Identifies with Sea Turtles

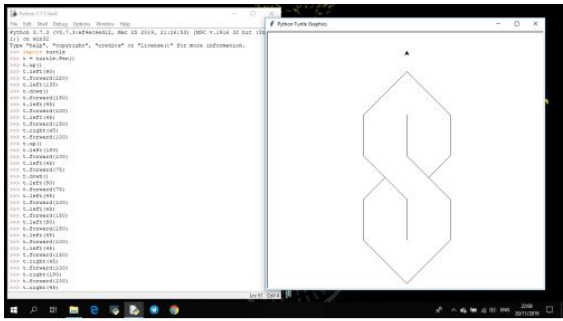
Aviva Weinbaum
Language Guru
Identifies with Rattlesnakes

Lindsey Weiskopf
System Architect
Identifies with Iguanas

About the Language

Language Overview & Motivation

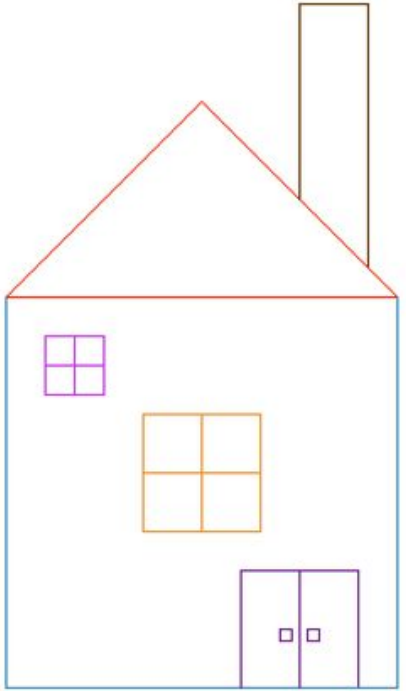
- Reptile is a programming language that is intended to support libraries that streamline the process of creating simply-coded graphics.
- Goal: to build upon the success of “beginner” programming languages, like Swift Playgrounds and Scratch, and libraries like Python Turtle to provide immediate gratification to the coders through graphics.



Cool S but made in Python Turtle Graphics because I have an IT class tomorrow

Key Language Features

- Java-like syntax
  - Strict-typing
  - Recursion
- Complex Types
  - RGB
  - Pointer
  - Canvas
- Built-in functions
- Production of PNG file



PLT or Architecture studio?

Complex Types

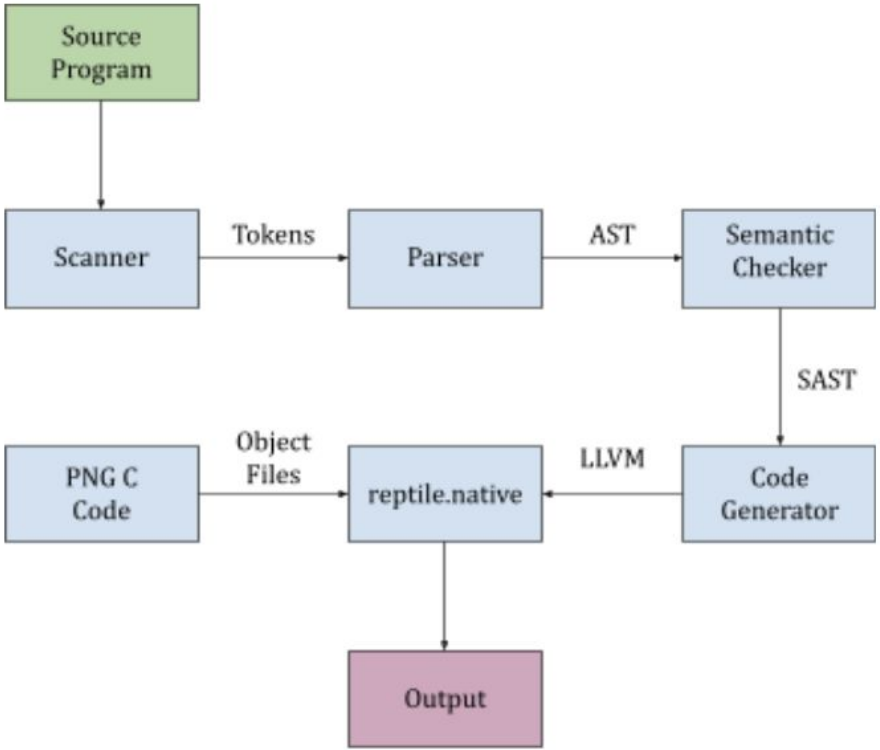
- **RGB** `(int r, int g, int b)`
  - Takes 3 int arguments to define color of pixels to be drawn
- **Pointer** `(int x, int y, struct rgb* color, float angle)`
  - Takes 2 int arguments to define starting position of pixel, 1 pointer to an Rgb struct to define color, 1 float argument to define an angle
- **Canvas** `(int x, int y)`
  - Takes 2 int arguments to define length and width dimensions of the PNG

Production of PNG

- `Libattopng C Library`
  - Used several functions to create, modify pixels, and save png, and clean up after:
    - `libattopng_new()`
    - `libattopng_set_pixel()`
    - `libattopng_save()`
    - `libattopng_destroy()`
- `Built-in functions`
    - `pixel()`
    - `save()`

About the Compiler

Architectural Design



Code

Code Generation

```
let rgb_t      = L.pointer_type(L.struct_type context
[| i32_t ; i32_t; i32_t  |]) in
 let pointer_t  = L.pointer_type(L.struct_type
context [| i32_t ; i32_t ; rgb_t ; float_t  |]) in
 let canvas_t   = L.pointer_type(L.struct_type
context [| i32_t ; i32_t  |]) in
            let pixelcons_t : L.lltype =
                L.function_type canvas_t  [| canvas_t ; rgb_t; i32_t ; i32_t |] in
            let pixelcons_fun : L.llvalue =
                L.declare_function "pixel" pixelcons_t the_module in

                                | SCall ("get_rgb_r", [rgb;]) ->
                                      let build_t : L.lltype =
                                        L.function_type i32_t  [|rgb_t;|] in
                                          let build_func : L.llvalue =
                                            L.declare_function "get_rgb_r" build_t the_module  in
                                        L.build_call build_func  [| expr builder locals rgb |]
                                          "get_rgb_r" builder
```

Struct Definitions & Constructor

```
struct Canvas {
   int x;
   int y;
   libattopng_t *png;
};

struct canvas* Canvas(int x, int y) {
   struct canvas *can = malloc(sizeof(struct canvas));
   can->x = x;
   can->y = y;
   can->png = libattopng_new(x, y, PNG_RGBA);
   return can;
}
```

Built-in Functions

```
#define RGBA(r, g, b, a) ((r) | ((g) << 8) | ((b) << 16) | ((a) << 24))

struct canvas* pixel(struct canvas* can, struct rgb* color, int x, int y)
{
   libattopng_set_pixel(can->png, x, y, RGBA(get_rgb_r(color) & 255,
get_rgb_g(color) & 255, get_rgb_b(color) & 255, (255 )));
   return can;
}

void save(struct canvas* can, char *filename) {
   libattopng_save(can->png, filename);
   libattopng_destroy(can->png);
}
```

Tortoise Library (.rt)

```
int xcur;
int ycur;
int tortup(Canvas can, Rgb color, int distance) {
   int counter = 0;
   while(counter < distance) {
       pixel(can, color, xcur, ycur-counter);
       counter = counter + 1;
   }
   ycur = ycur - distance;
   return 0;
}
int tortdown(Canvas can, Rgb color, int distance) {
   int counter = 0;
   while(counter < distance) {
       pixel(can, color, xcur, ycur+counter);
       counter = counter + 1;
   }
   ycur = ycur + distance;
   return 0;
}
```

```
int tortSE(Canvas can, Rgb color, float distance) {
   int counter = 0;
   float counter1 = 0.0;
   float step = distance * 0.707;
   while(counter1 < step) {
       pixel(can, color, xcur+counter, ycur+counter);
       counter1 = counter1 + 1.0;
       counter = counter + 1;
   }
   xcur = xcur + counter;
   ycur = ycur + counter;
   return 0;
}

int movetort(int x, int y) {
   xcur = x;
   ycur = y;
   return 0;
}
```

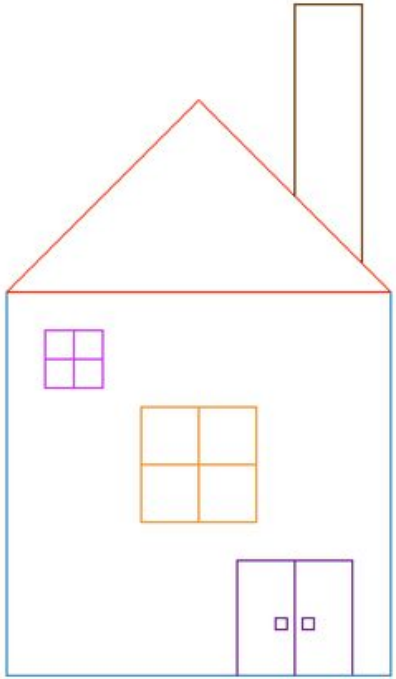
Demo

Demonstration of Coolest Reptile Program

~61,000 lines

Demonstration of the Second Coolest Reptile Program

featuring Tortoise



Conclusion

How did we get things to work?

- Several tests were used to test for the functionality of the basics (if/else statements, recursion, general arithmetic, scope, and more).

```
int main(){
    int i = 15;
    return i;
    i = 32;
}
```

Integration Testing

- Once we were able to generate code, we began to test the functionality of our structs, built-in functions, and creation of png files.

```
int main() {

Canvas can = Canvas (400, 400);

Rgb color = Rgb(0, 0, 0);

pixel(can, color, 200, 200);

save(can, “pixeltest.png”);

return 0;

}
```

Future Work

- Tortoise object
  - Object that serves as a visual Pointer (complex type) to indicate current positions and where pixels are being drawn
- Enable live drawing
  - Users can see the Tortoise object drawing each pixel once a command is completely typed
- User-defined Structs
  - Users can define their own object types and personalize the texture of the drawing, special effects, and more.

Questions?