Project Ίκαρος

Περιεχόμενα:

Εισαγωγή:

Ο «Ίκαρος» είναι project που δημιουργήθηκε με τη συνεργασία του κ. Οδυσσέα Κοντοβούρκη, καθηγητή του Τμήματος Αρχιτεκτονικής του Πανεπιστήμιου Κύπρου, και του Κωνσταντίνου Δουμανίδη, φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Αριστοτέλειου Πανεπιστήμιου Θεσσαλονίκης. Στόχος του Ίκαρου είναι να προσφέρει τις βάσεις για την εύκολη συγγραφή κώδικα που θα εξυπηρετεί τις εργασίες των φοιτητών και του ακαδημαϊκού προσωπικού του Τμήματος Αρχιτεκτόνων του Πανεπιστήμιου Κύπρου. Σε αρχική φάση ο Ίκαρος θα συμπεριλαμβάνει κώδικα γλώσσας Python που θα τρέχει πάνω στον επεξεργαστή αλγορίθμων Grasshopper του σχεδιαστικού περιβάλλοντος Rhinoceros. Ο Ίκαρος σχεδιάστηκε ώστε να αποτελείται από τέσσερα διαφορετικά αλλά αλληλένδετα μέρη: το παρόν εγχειρίδιο (που θα συμπεριλαμβάνει και κώδικα), τα κομμάτια κώδικα του εγχειριδίου σε ανεξάρτητα αρχεία τύπου .gh, τα αρχεία επέκτασης για το Grasshopper (.gha ή ghpy) για μια γκάμα χρήσιμων εργαλείων που δημιουργήθηκαν ειδικά για το Τμήμα Αρχιτεκτονικής αλλά και ένα διαδικτυακό εναποθετήριο ανοιχτού κώδικα στην πλατφόρμα Github. Σκοπός του Ίκαρου είναι να κατασκευαστεί με τρόπο που να προσφέρει τη δυνατότητα συνεχούς βελτίωση και εξέλιξη του, από την πανεπιστημιακή κοινότητα, πέραν της πρώτης φάσης της ανάπτυξης του.

Χρονοδιάγραμμα Ανάπτυξης:

Φάση 1η (Αύγουστος 2017)

Για την ανάπτυξη του Ίκαρου, τα πρώτα βήματα που ακολουθήθηκαν ήσαν τα εξής:

1. Συγγραφή του παρόν εγχειρίδιου με τα βασικά κομμάτια κώδικα.
2. Δημιουργία των αρχείων .gh για τον κώδικα.
3. Πακετάρισμα των εργαλείων που αναπτύχθηκαν ως επεκτάσεις του Grasshopper (.ghpy) χρησιμοποιώντας το Component SDK του Grasshopper Python.
4. Προετοιμασία του Github repository του Ίκαρου.

Φάση 2η (Απρίλιος 2018)

Οργάνωση υλικού Ίκαρου και δημοσίευση της 1ης έκδοσης (Icarus_v1.0) στο Github.

Βασική Σύνταξη της γλώσσας Python:

Το τμήμα αυτό του εγχειριδίου έχει σκοπό να κάνει μια γρήγορη εισαγωγή σε βασικά στοιχεία σύνταξης για κώδικα της γλώσσας Python ώστε ακόμα και όσοι δεν έχουν ιδιαίτερη επαφή με τον προγραμματισμό να μπορέσουν να κατανοήσουν, να επεξεργαστούν και να τροποποιήσουν τον κώδικα που παρέχει ο Ίκαρος για να συγγράψουν τα δικά τους εργαλεία.

Η Python είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου, δηλαδή τα προγράμματα που γράφονται σε Python είναι γενικώς φορητά (τρέχουν σε πολλά διαφορετικά συστήματα) και εύκολα στη συγγραφή και ανάγνωση. Είναι γλώσσα ανοιχτής φύσης (open source) και αυτή τη στιγμή βρίσκεται στην έκδοση 3.x.x.

Εκτέλεση εντολών: Οι εντολές στην Python εκτελούνται με τη σειρά που γράφτηκαν (με εξαίρεση την περίπτωση που καλούμε συναρτήσεις όπως θα δούμε αργότερα).

Σχόλια (Comments): Η Python επιτρέπει στον προγραμματιστή να τοποθετεί σχόλια μέσα στον κώδικα του για να γίνεται πιο ευανάγνωστος. Για να γράψουμε σχόλιο μιας γραμμής, απλά τοποθετούμε το χαρακτήρα `#` και μετά το σχόλιο μας. Για να γράψουμε σχόλιο έκτασης πολλαπλών γραμμών πληκτρολογούμε τους χαρακτήρες `'''` έπειτα το σχόλιο μας και στο τέλος για να κλείσουμε το σχόλιο μας ξανά τους χαρακτήρες `'''`. Συνήθως ο επεξεργαστής κειμένου για Python αυτόματα αλλάζει το χρώμα των σχολίων για να ξεχωρίζουν.

Εισαγωγή Βιβλιοθηκών (Importing Modules): Οι βιβλιοθήκες είναι αρχεία κώδικα που περιέχουν κάποιες έτοιμες λειτουργίες που μπορούμε να αξιοποιήσουμε στο πρόγραμμα μας. Για να εισάγουμε μια βιβλιοθήκη απλά γράφουμε **`import`** και το όνομα της βιβλιοθήκης που θέλουμε να εισάγουμε. Για τη συγγραφή κώδικα Python για τον επεξεργαστή αλγορύθμων Grasshopper εισάγουμε τις ακόλουθες δύο βιβλιοθήκες που μας επιτρέπουν να χρησιμοποιούμε κάποιες ειδικές εντολές του Grasshopper και του Rhino:

```
import rhinoscriptsyntax as rs
import ghpythonlib.components as ghc
```

Η rhinoscripsyntax μας επιτρέπει να χρησιμοποιούμε εντολές από την γλώσσα RhinoScript του περιβάλλοντος Rhino, ενώ η ghpytholib μας επιτρέπει να χρησιμοποιούμε εντολές του Grasshopper. Ανάλογα με τον σκοπό του προγράμματος μας εισάγουμε όποιο συνδυασμό των δύο κρίνουμε απαραίτητο.

Μεταβλητές (Variables): Στον προγραμματισμό οι μεταβλητές λειτουργούν σαν κουτιά που αποθηκεύουν δεδομένα. Στην Python έχουμε 5 είδη μεταβλητών: Αριθμούς (Numbers), Αλφαριθμητικά (Strings), Λίστες (Lists), Tuples και Χάρτες (Maps/Dictionaries). Για να δηλώσουμε μια μεταβλητή απλά γράφουμε το όνομα της και την εξισώνουμε με κάτι π.χ:

```
age = 19    #Δημιουργήσαμε τη μεταβλητή age και ισούται με 19

age = "Είναι 19"     #Τώρα αλλάξαμε τη μεταβλητή age και την εξισώσαμε με ένα αλφαριθμητικό
```

Σημείωση: Τα ονόματα μεταβλητών πρέπει οπωσδήποτε να αρχίζουν με γράμμα και να περιέχουν μόνο γράμματα, αριθμούς και το χαρακτήρα _ .

Μαθηματικοί τελεστές (Mathematical Operators): Για να επιδράσουμε σε δεδομένα στην Python χρησιμοποιούμε τους τελεστές +, -, *, /, //, % και **. Οι πρώτοι τέσσερεις είναι της πρόσθεσης, αφαίρεσης, πολλαπλασιασμού και διαίρεσης. Έπειτα έχουμε το // που λέγεται floor division, που κάνει διαίρεση και στρογγυλοποιεί το αποτέλεσμα προς τα κάτω (πχ. 7.5 // 2 = 3). Στη συνέχεια ο % λέγεται modulo και επιστρέφει το υπόλοιπο ακέραιας διαίρεσης (πχ: 5 % 2 = 1) ενώ ο ** κάνει εκθετικές πράξεις (π.χ 2**3 = 8). Η πρωτεραιότητα πράξεων είναι η ίδια με αυτή που μας είναι γνωστή από τα μαθηματικά (από τα δεξιά προς τα αριστερά όπως έχουμε παραθέσει τους τελεστές πιο πάνω). Παράδειγμα πράξεων:

```
num1 = 5
num2 = 6
num3 = num1 * num2 + 3          #Δηλαδή τώρα το num3 = 33
```

Σημείωση: Ο αριθμητικός τελεστής + μπορεί να χρησιμοποιηθεί για να ενώσει αλφαριθμητικά.

Σημείωση 2η: Στην Python αν βάλουμε ένα τελεστή και αμέσως μετά το = όπως εδώ (x+=5) είναι σαν να γράφουμε x = x+5.

Εκτύπωση στη κονσόλα (Output to the console): Αν θελήσουμε να εκτυπώσουμε πληροφορίες στο χρήστη μέσω της κονσόλας χρησιμοποιούμε την εντολή print() με παραμέτρους ό,τι θέλουμε να τυπώσουμε π.χ:

```
num1 = 5
print(num1)                                  #Θα εκτυπώσει: 5
print("Hello World")                         #Θα εκτυπώσει: Hello World
```

Αν θέλουμε να εκτυπώσουμε ένα συνδυασμό μεταβλητών, ανοίγουμε εισαγωγικά και βάζουμε με τη σειρά που θέλουμε να εμφανιστούν οι μεταβλητές, τα σύμβολα αντιστοίχησης τους. Το σύμβολο αντιστοίχισης για ακέραιους αριθμούς είναι %d, για αλφαριθμητικά το %s, για μεμονωμένους χαρακτήρες το %c και για δεκαδικούς αριθμούς το %f. Έπειτα ανοίγουμε παρένθεση με %() και παραθέτουμε με τη σειρά αυτή τις μεταβλητές που θα εκτυπώσουμε. Παράδειγμα:

```
num1 = 5
string1 = "Τα πρώτα"
```

```
float1 = 3.1415
print("%s %d στοιχεία του π είναι: %f" %(num1, string1, float1))
#Θα εκτυπώσει: Τα πρώτα 5 στοιχεία του π είναι: 3.1415
```

Αν θέλουμε κάπου μέσα σε ό,τι θα εκτυπώσουμε να αλλάξουμε γραμμή, απλά γράφουμε \n και εκεί αλλάζει η γραμμή π.χ: print("Hello World. \n I changed lines now!")

Αλφανουμερικά (Strings): Εδώ θα εξετάσουμε κάποιες λειτουργίες της Python που μας επιτρέπουν να χειριζόμαστε και να παίρνουμε πληροφορίες από αλφανουμερικές μεταβλητές.

```
string = "It's a beautiful day today"

string2 = string[0:4] #Εξισώνει το δεύτερο αλφανουμερικό με τους πρώτους 4 χαρακτήρες του πρώτου (από τον μηδενικό, μέχρι και το χαρακτήρα πριν το 4ο).

string2 = string[-5:] #Εξισώνει το δεύτερο αλφανουμερικό με τους τελευταίους 5 χαρακτήρες του πρώτου.

string2 = string[:-2] #Εξισώνει το δεύτερο αλφανουμερικό με τους χαρακτήρες του πρώτου από την αρχή μέχρι και πρίν τον προτελευταίο του.

string2 = string.capitalize()  #Εξισώνει το δεύτερο αλφανουμερικό με το πρώτο κάνοντας το πρώτο χαρακτήρα του κεφαλαίο.

string2 = string.find("today") #Εξισώνει το δεύτερο αλφανουμερικό με τη θέση της λέξης today στο πρώτο αλφανουμερικό.

string2 = string.isalpha()     #Εξισώνει το δεύτερο αλφανουμερικό με True αν το πρώτο αποτελείται μόνο από γράμματα (αλλιώς string2 = False).

string2 = string.isalnum()    #Εξισώνει το δεύτερο αλφανουμερικό με True αν το πρώτο αποτελείται από γράμματα και αριθμούς (αλλιώς string2 = False).

string2 = string.replace("today", "¯\_(ツ)_/¯")       #Εξισώνει το δεύτερο αλφανουμερικό με το πρώτο αντικαθιστώντας τη λέξη today με το ¯\_(ツ)_/¯. Σημειώστε πως το πρώτο αλφανουμερικό μένει ανέπαφο εδώ.

string2 = string.strip()      #Εξισσώνει το δεύτερο αλφανουμερικό με το πρώτο αφαιρώντας συγχρόνως τα κενά « ». Σημειώστε πως το πρώτο αλφανουμερικό μένει ανέπαφο εδώ.

string2 = len(string)         #Εξισώνει το δεύτερο αλφανουμερικό με το μήκος του πρώτου. Σημειώστε πως η ίδια συνάρτηση len() μπορεί να χρησιμοποιηθεί για να βρεθεί και το μήκος άλλων τύπων μεταβλητών π.χ λίστες, tuples, χάρτες κ.α
```

Λίστες (Lists): Ένας πολύ χρήσιμος τύπος μεταβλητής στην Python είναι η λίστα. Ας φανταστούμε μια λίστα ως ένα μονοδιάστατο πίνακα όπου η αρίθμηση των κελιών (δείκτες) ξεκινούν από το μηδέν και κάθε κελί μπορεί να περιέχει οποιουδήποτε τύπου δεδομένο. Κατ επέκτασην αυτής της σκέψης μπορούμε να δημιουργήσουμε μια λίστα της οποίας κάθε κελί είναι μια άλλη λίστα, φτιάχνοντας έτσι στην ουσία έναν πίνακα δύο διαστάσεων. Παρακάτω παρατίθενται οι λειτουργίες των λίστων:

```
list = [ 1, 2, 6, "Hello World", 9, 0]    #Δημιουργήσαμε μια λίστα με 6 στοιχεία

print(list)       #Μπορούμε να εκτυπώσουμε μια ολόκληρη λίστα με την εντολή print
```

```
list.append("Bananas", 88)    #Επεκτείνουμε τη λίστα προσθέτοντας 2 στοιχεία στο τέλος της.

num = list[2]     #Εξισώνουμε τη μεταβλητή num με το στοιχείο της λίστας που έχει δείκτη 2, δηλαδή το 6.

list2 = list[0:4]       #Εξισώνει τη δεύτερη λίστα με τα πρώτα 4 στοιχεία της πρώτης (από τον μηδενικό, μέχρι και το χαρακτήρα πριν το 4ο).

list3 = ["Grapes", "Apples", "Oranges"]
list.append(list3)
print(list[8][1])
#Δημιουργήσαμε μια νέα λίστα και την προσθέσαμε στο τέλος της αρχικής μας. Τώρα το τελευταίο στοιχείο της αρχικής μας λίστας (δηλαδή το list3) έχει δείκτη 8. Με την εντολή εκτύπωσης που γράψαμε, ζητάμε να εκτυπωθεί το αντικείμενο με δείκτη 1 της λίστας που βρίσκεται στον δείκτη 8 της αρχικής μας λίστας, άρα θα τυπωθεί η λέξη Apples.

list.insert(2, 99)      #Τοποθετήσαμε μέσα στη λίστα μας στο σημείο με δείκτη 2 την τιμή 99 και τώρα η λίστα μας είναι έτσι [1, 2, 99, 6, ...]

list.remove(99)         #Αφαιρέσαμε από τη λίστα μας το στοιχείο με την τιμή 99 και τώρα η λίστα μας είναι έτσι [1, 2, 6, ...]

del list[8] #Διαγράψαμε από τη λίστα μας το στοιχείο που βρισκόταν στον 8ο δείκτη

list = list + list3     #Ενώνει τις δύο λίστες παρόμοια με το append.

list.reverse()    #Αντιστρέψαμε τη λίστα, το τελευταίο στοιχείο έγινε πρώτο κ.ο.κ


list.sort()             #Οργανώνει τη λίστα κατά αύξουσα σειρά

print(len(list))        #Τυπώνει το μήκος της λίστας

print(max(list))        #Τυπώνει το μέγιστο στοιχείο της λίστας

print(min(list))        #Τυπώνει το ελάχιστο στοιχείο της λίστας
```

Λεξικά (Dictionaries): Ένας άλλος βασικός τύπος δεδομένων στην Python είναι τα λεξικά τα οποία μοιάζουν πολύ με τις λίστες ωστόσο τα στοιχεία που καταχωρούνται στα λεξικά αποτελούνται από δύο στοιχεία: τα δεδομένα και ένα μοναδικό κλειδί που αντιστοιχεί σε αυτά τα δεδομένα. Παράδειγμα: Πινακίδα αυτοκινήτου (κλειδί) – Μοντέλο αυτοκινήτου (δεδομένα).

```
map = {"JBO 007" : "Aston Martin DB5"
       "ASR 879" : "Mazda Demio"
       "LOC 346" : "Toyota Yaris"}  #Δημιουργία χάρτη

print(map["JBO 007"])   #Τυπώνει τα δεδομένα που αντιστοιχούν στο κλειδί JBO 007

del map["LOC 346"]      #Διαγράφει τη καταχώρηση με κλειδί LOC 346

map["ASR 879"] = "Tesla Model 3S"   #Αλλάζει τη καταχώρηση με κλειδί ASR 879

len(map)                #Τυπώνει τον αριθμό των καταχωρήσεων

string = map.get("JBO 007")   #Εξισώνει το string με τα δεδομένα του JBO 007

print(map.keys)         #Τυπώνει μια λίστα με τα κλειδιά του χάρτη
```

```
print(map.values)        #Τυπώνει μια λίστα με τα δεδομένα του χάρτη

print(map)               #Τυπώνει ολόκληρο το χάρτη
```

Λογικοί τελεστές (Logical Operators): Προκειμένου να γράψουμε λογικές εκφράσεις και συνθήκες στο προγραμματισμό χρειαζόμαστε τους λογικούς τελεστές. Στην Python έχουμε τους ακόλουθους λογικούς τελεστές:

```
==          #Εξετάζει ισότητα μεταξύ δύο μεταβλητών π.χ num == 5, αν num όντως
ισούται με 5 τότε επιστρέφει True αλλιώς επιστρέφει False.


!=          #Εξετάζει την ανισότητα δύο μεταβλητών π.χ num == 3, αν num είναι
άνισο του 3 τότε επιστρέφει True αλλιώς επιστρέφει False.

>           #Εξετάζει αν η μεταβλητή στα αριστερά είναι μεγαλύτερη από αυτή στα
δεξιά και επιστρέφει True αν ισχύει, αλλιώς επιστρέφει False.

<           #Εξετάζει αν η μεταβλητή στα αριστερά είναι μικρότερη από αυτή στα
δεξιά και επιστρέφει True αν ισχύει, αλλιώς επιστρέφει False.

>=          #Εξετάζει αν η μεταβλητή στα αριστερά είναι μεγαλύτερη ή ίση με αυτή
στα δεξιά και επιστρέφει True αν ισχύει, αλλιώς επιστρέφει False.

<=          #Εξετάζει αν η μεταβλητή στα αριστερά είναι μικρότερη ή ίση με αυτή
στα δεξιά και επιστρέφει True αν ισχύει, αλλιώς επιστρέφει False.

and         #Ο τελεστής αυτός τοποθετείται ανάμεσα σε μεταβλητές τύπου True/False
ή σε συνθήκες που επιστρέφουν True/False. Αν και οι δύο συνθήκες που βρίσκονται
αριστερά και δεξιά του τελεστή and είναι True, τότε επιστρέφει True, αλλιώς
επιστρέφει False. Π.χ: Έστω num = 5, num2 = 3 τότε η παρένθεση ((num == 5) and
(num2 == 3)) θα επιστρέψει True, ενώ η παρένθεση ((num != 5) and (num2 == 3)) θα
επιστρέψει False.

or          #Ο τελεστής or λειτουργεί παρόμοια με τον and με τη διαφορά ότι δεν
χρειάζεται και οι δύο συνθήκες στα αριστερά και δεξιά του να είναι αληθείς για να
επιστρέψει True, αρκεί μόνο μία από τις δύο τους να είναι. Π.χ: Έστω num = 5,
num2 = 3 τότε η παρένθεση ((num == 5) or (num2 == 3)) θα επιστρέψει True και η
παρένθεση ((num != 5) or (num2 == 3)) θα επιστρέψει πάλι True.

not         #Ο τελεστής αυτός τοποθετείται μπροστά από μια συνθήκη και
αντιστρέφει ό,τι επιστρέψει η συνθήκη π.χ: Έστω num = 5, η παρένθεση (not(num ==
5)) θα επιστρέψει False και η (not(not(num == 5))) θα επιστρέψει True.
```

Δομές απόφασης (if Statement): Προκειμένου να ελέγξουμε τη ροή του προγράμματος μας χρειαζόμαστε δομές απόφασης, στις οποίες αν ικανοποιείται μια ορισμένη συνθήκη εκτελείται ένα σετ εντολών. Στην python έχουμε μία βασική τέτοια δομή, την if-elif-else. Στη δομή αυτή, αρχικά εξετάζεται η συνθήκη της if, αν η συνθήκη είναι αληθής, εκτελείται το μπλοκ εντολών της if, αν όχι, εξετάζονται οι συνθήκες των elif (για κάθε if μπορούμε να έχουμε από μηδέν εώς άπειρα elif) και αν ικανοποιηθεί μια από τις συνθήκες αυτές, τρέχει το αντίστοιχο μπλοκ εντολών. Αν πάλι όμως δεν ικανοποιηθεί κάποια από αυτές τις συνθήκες τρέχει το μπλοκ εντολών της else η οποία δεν έχει συνθήκη (για κάθε if μπορούμε να έχουμε από καμία εώς μία else). Το μπλοκ εντολών που θέλουμε

να εκτελέσουμε σε κάθε περίπτωση το καθορίζουμε γράφοντας το με ένα διάστημα tab προς τα μέσα. Παράδειγμα:

```
if(age >= 18)
      print("Μπορείς να πιείς και να οδηγήσεις (όχι όμως μ αυτή τη σειρά)")
elif((age < 18) and (age > 17))
      print("Μπορείς να οδηγήσεις με τη συνοδεία ενήλικα")
else
      print("Δεν μπορείς ούτε να οδηγήσεις ούτε να πιείς")
```

Δομές επανάληψης (Looping): Άλλο χρήσιμο εργαλείο που μας προσφέρει η Python είναι οι δομές επανάληψης οι οποίες τρέχουν όσο η συνθήκη που έχουμε θέσει είναι αληθής. Συγκεκριμένα η Python έχει δύο δομές επανάληψης: την for και την while. Η πρώτη χρησιμοποιείται συνήθως για όταν ξέρουμε εξαρχής πόσες φορές θέλουμε να τρέξει το μπλοκ εντολών της ενώ η δεύτερη είναι για όταν δεν ξέρουμε, κανείς όμως δεν μας περιορίζει στο να χρησιμοποιήσουμε όποια από τις δύο μας φαίνεται πιο βολική στην εφαρμογή μας.

Εφαρμογές για επαναληπτική δομή for:

```
for x in range(0, 10)
      print(x ," ", end="")

#Οι δύο αυτές γραμμές δημιουργούν ένα βρόγχο επανάληψης που τρέχει 10 φορές και το x παίρνει τιμές από το 0 εώς το 9. Η εντολή print που γράψαμε για κάθε επανάληψη (κάθε φορά που τρέχει το μπλοκ εντολών) τυπώνει το x, αφήνει ένα κενό (παράμετρος " ") και δεν αλλάζει γραμμή (παράμετρος end=""). Η έξοδος είναι ως εξής: 0 1 2 3 4 5 6 7 8 9.

for x in list
      print(x)

#Οι δύο αυτές γραμμές δημιουργούν ένα βρόγχο επανάληψης που τρέχει όσες φορές όσα τα στοιχεία της λίστας και κάθε φορά που τρέχει ο βρόγχος επανάληψης το x παίρνει την τιμή ενός στοιχείου της λίστας και χρησιμοποιώντας την εντολή print το τυπώνουμε.

for x in [1 2 3 5 8 13]
      print(x)

#Οι δύο αυτές γραμμές δημιουργούν ένα βρόγχο επανάληψης που τρέχει 6 φορές και κάθε φορά που τρέχει ο βρόγχος επανάληψης το x παίρνει την τιμή του επόμενου στοιχείου της λίστας [1 2 3 5 8 12] και χρησιμοποιώντας την εντολή print το τυπώνουμε.
```

Εφαρμογές για επαναληπτική δομή while:

```
i = 0;
while (i <= 20):
    if(i == 5):
        i = i+1
        continue
    if(i%2 == 1):
        print(i)
    elif(i == 14):
        break
    i += 1
```

#Στις γραμμές αυτές αφού δηλώσουμε μια μεταβλητή – δείκτη (το i), φτιάχνουμε ένα βρόγχο while ο οποίος τρέχει όσο το i είναι μικρότερο ή ίσο του 20 (θα τρέξει δηλαδή 21 φορές ο βρόγχος). Μετά ελέγχουμε αν το i είναι ίσο με το 5, αν είναι, αυξάνουμε το i κατά 1 και μετά με την εντολή continue, δεν εκτελούμε το υπόλοιπο μπλοκ εντολών αλλά λέμε στον υπολογιστή να επανεξετάσει την αρχική συνθήκη του βρόγχου επανάληψης και να συνεχίσει από εκεί. Στο δεύτερο if ελέγχουμε αν το i είναι μονός αριθμός και αν είναι τον τυπώνουμε. Στη συνέχεια, στο elif εξετάζουμε αν το i είναι ίσο με 14, αν είναι λέμε στον υπολογιστή να βγεί από το βρόγχο επανάληψης και να συνεχίσει το πρόγραμμα χρησιμοποιώντας την εντολή break. Στο τέλος του μπλοκ εντολών λέμε να ανεβάσει την τιμή του i κατά 1. Το πρόγραμμα μας δηλαδή θα τυπώσει όλους τους μονούς αριθμούς από το 0 εώς το 14 προσπερνώντας το 5. Σε τέτοιους βρόγχους επανάληψης είναι σημαντικό να προσέχουμε να μην υπάρξει πιθανότητα ο δείκτης μας να μένει σταθερός, διότι σε αυτή τη περίπτωση θα είχαμε έναν ατέρμονο βρόγχο και η εκτέλεση του προγράμματος μας θα κολούσε εκεί.

Συναρτήσεις (Functions): Η συγγραφή συναρτήσεων αποτελεί μια από τις πιο χρήσιμες δυνατότητες της Python καθότι μας επιτρέπει να κάνουμε τον κώδικα μας πιο μικρό και ευανάγνωστο αφού μπορούμε να «ανακυκλώνουμε» κώδικα. Η συγγραφή συναρτήσεων στην Python γίνεται πάντα στο πάνω μέρος του προγράμματος μας ως εξής: def όνομα_συνάρτησης(Ορίσματα): μπλοκ εντολών και μετά η εντολή return με τη τιμή που θέλουμε να επιστρέψουμε. Παράδειγμα:

```
def rad2degrees(rad):
      degree = 57.29 * rad
      return degree

num1 = 6
num2 = rad2degree(num1)
```

#Στο παραπάνω παράδειγμα δηλώνουμε μια συνάρτηση rad2degree που μετατρέπει τα ακτίνια σε μοίρες και έχει ως όρισμα τα ακτίνια που μπαίνουν μέσα στη μεταβλητή rad. Στο κυρίως πρόγραμμα δημιουργούμε μια μεταβλητή num1 με τιμή 6 και μετά δημιουργούμε μια μεταβλητή num2 που λέμε πως είναι ίση με τη τιμή που επιστρέφει η συνάρτηση rad2degree αν δώσουμε σαν όρισμα το num1. Επομένως η ροή του προγράμματος ανεβαίνει πάνω στη συνάρτηση, το rad γίνεται 6 και μετατρέπουμε τα rad σε ακτίνια με το γνωστό τύπο, έπειτα με την εντολή return degree η συνάρτηση επιστρέφει τη επιθυμητή τιμή και την τοποθετεί μέσα στη μεταβλητή num2. Σημειώστε πως οι μεταβλητές rad και degree είναι τοπικές μεταβλητές της συνάρτησης και δεν είναι προσβάσιμες έξω από αυτήν.

Κλάσεις και Αντικείμενα (Classes and Objects): Η Python θεωρείται ιδιαίτερα ευέλικτη γλώσσα προγραμματισμού διότι μεταξύ άλλων επιτρέπει στους χρήστες να προγραμματίζουν με κλάσεις (classes) και αντικείμενα (objects), είναι δηλαδή γλώσσα συμβατή με τις αρχές του αντικειμενοστρφούς προγραμματισμού (object-oriented programming). Ας φανταστούμε μια κλάση ως τη συνταγή και τα αντικείμενα ως ό,τι φτιάχνουμε με τη βοήθεια της συνταγής. Μέσα σε ένα πρόγραμμα μπορούμε να έχουμε πολλαπλές κλάσεις και πολλαπλά ανεξάρτητα μεταξύ τους αντικείμενα από τη κάθε κλάση. Μια κλάση λοιπόν μπορεί να περιέχει μέσα της μια συλλογή από μεταβλητές και μεθόδους (συναρτήσεις). Αν δημιουργήσουμε ένα αντικείμενο από μια κλάση, το αντικείμενο αυτό περιέχει όλες αυτές τις μεταβλητές και συναρτήσεις και μπορούμε με τη βοήθεια αντικειμένων να απλοποιήσουμε και να λύσουμε πλήθος προβλημάτων. Στην python για να αποκτήσουμε πρόσβαση στα στοιχεία ενός αντικειμένου χρησιμοποιούμε τον τελεστή «.» π.χ:

```
object.variable = 1           #Αλλάζει τη μεταβλητή variable του αντικειμένου
object.function()       #Εκτελεί την συνάρτηση function του αντικειμένου
```

Όταν προγραμματίζουμε για το Grasshopper συνήθως δεν χρειάζεται να δημιουργήσουμε δικά μας αντικείμενα ωστόσο η παραπάνω εισαγωγή μπορεί να εξηγήσει πως να χειριζόμαστε κάποια αντικείμενα π.χ αν φτιάξουμε ένα τόξο από 3 σημεία στο Grasshopper, το αντικείμενο μας θα εμπερικλείει μεταξύ άλλων και την ιδιότητα της ακτίνας, την οποία για να την δούμε μπορούμε να τυπώσουμε:

```
print(arcObject.radius)
```

Αν όμως επιθυμούμε να δημιουργήσουμε μια δική μας κλάση, χρειάζεται να γράψουμε οπωσδήποτε μια συνάρτηση αρχικών συνθηκών. Αυτή η συνάρτηση εκτελείται όταν δημιουργούμε ένα αντικείμενο και εκτελεί μια σειρά ενεργειών που θέλουμε εμείς (συνήθως τη χρησιμοποιούμε για να θέτει αρχικές τιμές σε μια σειρά από μεταβλητές). Παράδειγμα:

```
class Human:
      def __init__(self, param1, param2, param3, param4, param5):
            self.name = param1      #Η συνάρτηση αρχικών συνθηκών δηλώνεται με τη
            self.age = param2       #γραφή def __init__(παράμετροι). Η πρώτη
            self.gender = param3    #παράμετρος είναι η self που στην ουσία είναι
            self.height = param4    #το ίδιο το αντικείμενο. Οι μεταβλητές
            self.energy = param5    # self.x είναι οι μεταβλητές της κλάσης.

      def sleep(self):
            self.energy = self.energy + 30

      def work(self):
            self.energy = self.energy – 40

      def introduce(self):
            print(self.name)

      fav_song = “Never gonna give you up – Rick Astley”


human_Maria = Human(“Maria”, 23, “female”, 1.72, 100)
#Δημιουργήσαμε ένα αντικείμενο από τη κλάση Human

human_Maria.work()      #Εκτελέσαμε τη μέθοδο work του αντικείμενου

print(human_Maria.energy)     #Τυπώσαμε την μεταβλητή energy του αντικειμένου μας
```

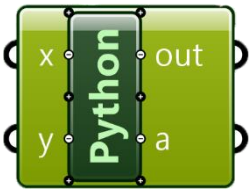
Εδώ τελειώνει το τμήμα αυτό του εγχειρίδιου, ωστόσο αν επιθυμεί ο αναγνώστης να μελετήσει τη Python σε μεγαλύτερο βάθος προτείνουμε να δει τη σελίδα learnpython.org και αν θελήσει να έχει ένα καλό cheat sheet προτείνουμε αυτό από το New Think Tank.

1. http://learnpython.org/en/Classes_and_Objects
2. http://www.newthinktank.com/2014/11/python-programming/

Κώδικας:

Σκοπός αυτού του μέρους του Ίκαρου είναι να προσφέρει κομμάτια κώδικα που θα πραγματοποιούν λειτουργίες των βασικών εργαλείων που είναι ήδη διαθέσιμα στο Grasshopper δίνοντας στο χρήστη τη δυνατότητα να τα τροποποιήσει και να τα συνδυάσει δημιουργώντας δικά του πιο σύνθετα εργαλεία. Ο κώδικας που ακολουθεί, για καλύτερη οργάνωση, παρατίθεται με την κατηγοριοποίηση που συναντά ο χρήστης στο περιβάλλον του Grasshopper.

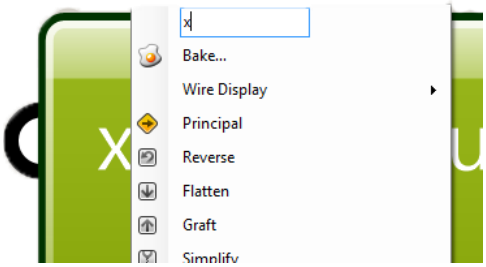
Εισαγωγή στο Python component του Grasshopper: Για να έχουμε τη δυνατότητα να γράψουμε δικό μας κώδικα μέσα στο περιβάλλον του Grasshopper χρειάζεται να εγκαταστήσουμε την επέκταση GHPYTHON για το Grasshopper από τη σελίδα Food4Rhino. Μετά κάτω από το μενού Maths -> Script θα έχουμε ένα νέο component με το όνομα Python Script το οποίο όταν το τοποθετήσουμε στον καμβά μας θα μοιάζει κάπως έτσι:



Στην αριστερή μεριά βρίσκονται οι μεταβλητές εισόδου ενώ στα δεξιά οι μεταβλητές εξόδου. Μόλις τοποθετήσουμε το component στον καμβά έχει δύο μεταβλητές εισόδου (x και y) και δύο εξόδου (out και a).

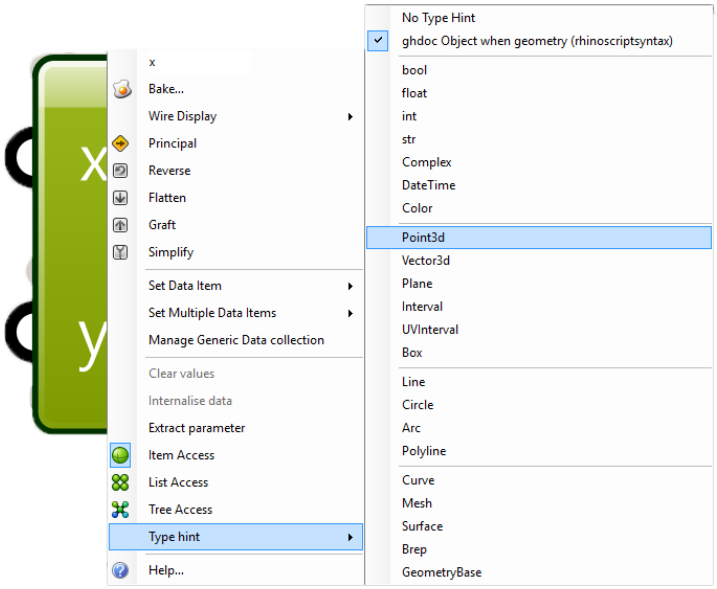
Αν θέλουμε να προσθέσουμε ή να αφαιρέσουμε μεταβλητές εισόδου και εξόδου απλά κάνουμε ζουμ προς το component και από όποια μεριά επιθυμούμε πατάμε το σημαδάκι + ή – για να προσθέσουμε ή να αφαιρέσουμε την επιθυμητή μεταβλητή.

Μέσα στο πρόγραμμα μας, τα ονόματα των μεταβλητών εισόδου και εξόδου μας πρέπει να συμφωνούν με αυτά πάνω στο component. Για να μετονομάσουμε μια μεταβλητή πάνω στο component κάνουμε δεξί κλίκ πάνω της και χρησιμοποιούμε την πρώτη επιλογή.

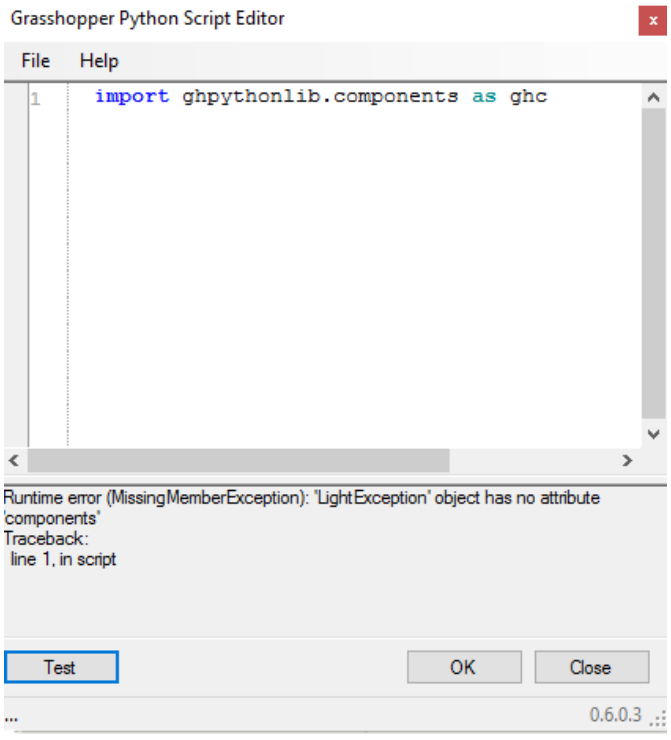


Μια καλή πρακτική που ακολουθούμε όταν προγραμματίζουμε με τη Python στο Grasshopper είναι αφού θέσουμε τις μεταβλητές μας, να πάμε και να κάνουμε δεξί κλικ πάνω τους και κάτω από το

μενού Type Hint επιλέγουμε το είδος της μεταβλητής που έχουμε σκοπό να εισάγουμε στο component. Επιπλέον αν μία από τις εισόδους μας είναι λίστα αντικειμένων, από Item Access αλλάζουμε την επιλογή σε List Access όταν κάνουμε δεξί κλικ πάνω στην είσοδο. Αυτό βοηθάει το Grasshopper να επεξεργάζεται με μεγαλύτερη ακρίβεια τα δεδομένα που εισάγονται στο πρόγραμμα.



Τέλος κάνοντας διπλό κλικ πάνω στο component ανοίγει ο επεξεργαστής κώδικα όπου μέσα γράφουμε το πρόγραμμα μας στη Python. Αφού έχουμε τελειώσει με τη συγγραφή του κώδικα μπορούμε να πατήσουμε το κουμπί στο κάτω αριστερό μέρος του παραθύρου με την ετικέτα Test για να δοκιμάσει το Grasshopper αν ο κώδικας μας είναι εκτελέσιμος. Κάτω από τον επεξεργαστή υπάρχει η κονσόλα η οποία δείχνει την έξοδο από τον έλεγχο του κώδικα αλλά και χρήσιμες πληροφορίες για διάφορες συναρτήσεις του Grasshopper την ώρα που γράφουμε το κώδικα.



Περιεχόμενα κώδικα:

- Vector
  - Vector
    - *Vector (X, Y, Z)*
    - *Vector (από δύο σημεία)*

- Curve
  - Analysis
    - End Points
    - Evaluate Curve
  - Division
    - Divide Curve
    - Contour
  - Primitive
    - Circle
      - *Circle (από ακτίνα και επίπεδο)*
      - *Circle (από τρία σημεία)*
      - *Circle (από σημείο, διάνυσμα και ακτίνα)*
      - *Circle (που να ικανοποιεί πλήθος σημείων)*
      - *Circle (που να εφάπτεται σε δύο καμπύλες)*
      - *Circle (που να εφάπτεται σε τρείς καμπύλες)*
    - Arc
      - *Arc (από επίπεδο, ακτίνα και γωνία)*
      - *Arc (από αρχικό, ενδιάμεσο και τελικό σημείο)*
      - *Arc (από αρχικό και τελικό σημείο και εφαπτόμενο διάνυσμα)*
  - Spline
    - Nurbs Curve
    - Interpolate
  - Util
    - Explode
    - Join Curves

- Surface
  - Analysis
    - Deconstruct Brep
    - Volume
  - Freeform
    - 4Point Surface
    - Edge Surface
  - Primitive
    - Bounding Box

- Sphere
- Util
  - Divide Surface
  - Isotrim
- Intersect
  - Mathematical
    - Curve | Line
  - Physical
    - Curve | Curve
- Transform
  - Affine
    - Scale
  - Array
    - Linear Array
  - Euclidean
    - Move
    - Orient
    - Rotate 3D
  - Morph
    - Box Morph
- Surface Box

Params:

Geometry:

Point:

Ο παρακάτω κώδικας δημιουργεί ένα παραμετροποιήσιμο σημείο στο τρισδιάστατο χώρο στο Rhino. Οι παράμετροι x,y,z μπορούν να οριστούν από άλλα μέλη του Grasshopper (sliders, control knobs κλπ) ή να οριστούν από τον χρήστη μέσα στον κώδικα του.

Μεταβλητές εισόδου:

x – Άξονας x y – Άξονας y z – Άξονας z

Μεταβλητές εξόδου:

Point – Σημείο που δημιουργείται

```
import rhinoscriptsyntax as rs
import ghpythonlib.components as ghc

'''Αφαιρέστε το # από τις παρακάτω γραμμές για να προκαθορίσετε τις τιμές
x,y,z'''

#x = 0
#y = 0
#z = 0

Point = ghc.ConstructPoint(x,y,z)
```

Maths:

Domain:

Construct Domain (από 2 αριθμούς):

Ο παρακάτω κώδικας δημιουργεί ένα πεδίο (domain) από δύο αριθμούς.

Μεταβλητές εισόδου:

number1 – Πρώτος αριθμός number2 – Δεύτερος αριθμός

Μεταβλητές εξόδου:

Domain – Τομέας που δημιουργείται

```
import ghpythonlib.components as ghc

#Αλλάξτε τα number1, number2 πιο κάτω για να προκαθορίσετε τις τιμές τους

Domain = ghc.ConstructDomain(number1, number2)
```

Construct Domain^2 (από 2 ζευγάρια αριθμών):

Ο παρακάτω κώδικας δημιουργεί ένα δισδιάστατο πεδίο (domain) από δύο ζευγάρια αριθμών.

Μεταβλητές εισόδου:

pair1min – Ελάχιστος αριθμός 1ης διάστασης pair1max – Μέγιστος 1ης διάστασης

pair2min – Ελάχιστος αριθμός 2ης διάστασης pair2max – Μέγιστος 2ης διάστασης

Μεταβλητές εξόδου:

Domain – Τομέας που δημιουργείται

```
import ghpythonlib.components as ghc

'''Αλλάξτε τα pair1min, pair1max, pair2min, pair2max πιο κάτω για να
προκαθορίσετε τις τιμές τους'''

Domain = ghc.ConstructDomain2(pair1min, pair1max, pair2min, pair2max)
```

Remap:

Ο παρακάτω κώδικας δέχεται δύο πεδία αριθμών και έναν αριθμό μέσα στο πρώτο πεδίο και σαν έξοδο δίνει τον αριθμό αυτό επανατοποθετημένο μέσα στο δεύτερο πεδίο με αποκοπή ή χωρίς.

Μεταβλητές εισόδου:

number – Αριθμός που ανήκει στο πρώτο πεδίο

sourceDomain– Πεδίο στο οποίο ανήκει ο number

targetDomain – Πεδίο στο οποίο επανατοποθετείται ο αριθμός

Μεταβλητές εξόδου:

RemappedNum – Ο επανατοποθετημένος αριθμός

ClippedNum – Ο επανατοποθετημένος αριθμός με αποκοπή

```
import ghpythonlib.components as ghc

RemappedNum = ghc.RemapNumbers(number, sourceDomain, targetDomain).mapped
ClippedNum = ghc.RemapNumbers(number, sourceDomain, targetDomain).clipped
```

Sets:

Lists:

List Item:

Ο παρακάτω κώδικας δέχεται μια λίστα, έναν δείκτη και μια μεταβλητή bool και εκτυπώνει το αντικείμενο της λίστας με τον αριθμοδείκτη που δώσαμε με ή χωρίς τον δείκτη (αυτό ρυθμίζεται από τη μεταβλητή τύπου bool).

Μεταβλητές εισόδου:

index – Δείκτης wrap – Η μεταβλητή τύπου bool list - Λίστα

Μεταβλητές εξόδου:

Item – Το αντικείμενο που αναζητούμε

```
import ghpythonlib.components as ghc

'''Αλλάξτε το index σε αριθμό και το wrap σε true ή false αν επιθυμείτε να προκαθορίσετε τις τιμές τους'''

item = ghc.ListItem(list,index,wrap)
```

List Length:

Ο παρακάτω κώδικας δέχεται μια λίστα και ως έξοδο δίνει το μήκος της.

Μεταβλητές εισόδου:

list - Λίστα

Μεταβλητές εξόδου:

Length – Το μήκος της λίστας

```
import ghpythonlib.components as ghc

item = ghc.ListLength(list)
```

Sequence:

Cull Pattern:

Ο παρακάτω κώδικας δέχεται μια λίστα και ένα μοτίβο (λίστα αριθμών που αποτελείται από 0 και 1) σύμφωνα με το οποίο αφαιρεί στοιχεία από τη λίστα εισόδου και δίνει σαν έξοδο τη νέα λίστα που προκύπτει.

Μεταβλητές εισόδου:

listIn– Λίστα εισόδου pattern – Λίστα μοτίβου

Μεταβλητές εξόδου:

listOut – Λίστα εξόδου

```
import ghpythonlib.components as ghc

ListOut = ghc.CullPattern(listIn, pattern)
```

Sequence:

Ο παρακάτω κώδικας δέχεται σαν είσοδο ένα σετ χαρακτήρων, το επιθυμητό μήκος και προαιρετικά παραμέτρους για τη μορφοποίηση που θέλουμε να έχει η ακολουθία της εξόδου.

Μεταβλητές εισόδου:

length – Μήκος ακολουθίας characters – Λίστα χαρακτήρων

format – Παράμετροι μορφοποίησης

Μεταβλητές εξόδου:

SequenceOut – Ακολουθία

```
import ghpythonlib.components as ghc

SequenceOut = ghc.Sequence(length, characters, format)
```

Series:

Ο παρακάτω κώδικας δημιουργεί μια αριθμητική σειρά με βάση έναν αρχικό αριθμό, το βήμα και το μήκος της επιθυμητής σειράς.

Μεταβλητές εισόδου:

start – Αρχή σειράς step – Βήμα σειράς length – Μήκος σειράς

Μεταβλητές εξόδου:

SeriesOut – Σειρά που δημιουργείται

```
import ghpythonlib.components as ghc

'''Αντικαταστήστε τα start, step και length για να προκαθορίσετε τις τιμές τους'''

SeriesOut = ghc.Series(start, step, length)
```

Vector:

Grid:

Rectangular:

Ο παρακάτω κώδικας δημιουργεί ένα ορθογώνιο πλέγμα χρησιμοποιώντας ένα επίπεδο, το επιθυμητό μήκος στον x και y άξονα και το πόσα κελιά επιθυμεί ο χρήστης να έχει σε κάθε άξονα. Οι παράμετροι sizeX, sizeY, cellsX, cellsY μπορούν να οριστούν από άλλα μέλη του Grasshopper (sliders, control knobs κλπ) ή να οριστούν από τον χρήστη μέσα στον κώδικα του.

Μεταβλητές εισόδου:

sizeX – Μήκος στο x άξονα sizeY – Μήκος στον y άξονα

cellsX – Αριθμός κελίων στον x άξονα cellsY – Αριθμός κελιών στον y άξονα

plane – Επίπεδο

Μεταβλητές εξόδου:

Cells – Λίστα των κελιών του πλέγματος Points – Λίστα των σημείων του πλέγματος

```
import ghpythonlib.components as ghc

'''Αντικαταστήστε τα sizeX, sizeY, cellsX, cellsY αν επιθυμείτε να προκαθορίσετε τις τιμές τους'''
```

```
Cells = ghc.Rectangular(plane, sizeX, sizeY, cellsX, cellsY).cells
Points = ghc.Rectangular(plane, sizeX, sizeY, cellsX, cellsY).points
```

Plane:

Construct Plane:

Ο παρακάτω κώδικας χρησιμοποιεί σαν είσοδο ένα σημείο και δύο διανύσματα (ένα για το x άξονα και ένα για τον y άξονα) και δημιουργεί ένα επίπεδο. Το σημείο του επιπέδου αλλά και τα διανύσματα μπορούν να οριστούν μέσω του περιβάλλοντος του Grasshopper ή χρησιμοποιώντας κώδικα (βλέπε Vector -> Point -> Construct Point και Vector -> Vector -> Vector αντίστοιχα).

Μεταβλητές εισόδου:

xVector – Διάνυσμα του x άξονα yVector– Διάνυσμα του y άξονα

origin – Σημείο για ορισμό του επιπέδου

Μεταβλητές εξόδου:

Plane – Επίπεδο που δημιουργείται

```
import ghpythonlib.components as ghc

Plane = ghc.ConstructPlane(origin, xVector, yVector)
```

Point:

Construct Point:

Ο παρακάτω κώδικας δημιουργεί ένα παραμετροποιήσιμο σημείο στο τρισδιάστατο χώρο στο Rhino. Οι παράμετροι x,y,z μπορούν να οριστούν από άλλα μέλη του Grasshopper (sliders, control knobs κλπ) ή να οριστούν από τον χρήστη μέσα στον κώδικα του.

Μεταβλητές εισόδου:

x – Άξονας x y – Άξονας y z – Άξονας z

Μεταβλητές εξόδου:

Point – Σημείο που δημιουργείται

```
import rhinoscriptsyntax as rs
import ghpythonlib.components as ghc

'''Αφαιρέστε το # από τις παρακάτω γραμμές για να προκαθορίσετε τις τιμές
x,y,z'''

#x = 0
#y = 0
#z = 0

Point = ghc.ConstructPoint(x,y,z)
```

Distance:

Ο παρακάτω κώδικας δέχεται σαν είσοδο δύο σημεία στο χώρο και υπολογίζει τη μεταξύ τους απόσταση. Τα σημεία μπορούν να οριστούν μέσα από το περιβάλλον του Grasshopper ή από τον κώδικα του χρήστη (δες Vector -> Point -> Construct Point).

Μεταβλητές εισόδου:

pointA – Πρώτο σημείο pointB – Δεύτερο σημείο

Μεταβλητές εξόδου:

DistanceOut – Απόσταση σημείων

```
import ghpythonlib.components as ghc

DistanceOut = ghc.Distance(pointA, pointB)
```

Vector:

Vector XYZ:

Ο παρακάτω κώδικας δημιουργεί ένα διάνυσμα με παραμέτρους x, y, z. Οι παράμετροι x,y,z μπορούν να οριστούν από άλλα μέλη του Grasshopper (sliders, control knobs κλπ) ή να οριστούν από τον χρήστη μέσα στον κώδικα του.

Μεταβλητές εισόδου:

x – Παράμετρος x y – Παράμετρος y z – Παράμετρος z

Μεταβλητές εξόδου:

Vector – Διάνυσμα που δημιουργείται VectorMagnitude – Μέτρο του διανύσματος

```
import rhinoscriptsyntax as rs
import ghpythonlib.components as ghc

'''Αφαιρέστε το # από τις παρακάτω γραμμές για να προκαθορίσετε τις τιμές
x,y,z'''

#x = 0
#y = 0
#z = 0

Vector = ghc.VectorXYZ(x,y,z).vector
VectorMagnitude = ghc.VectorLength(Vector)
```

Vector από 2 σημεία:

Ο παρακάτω κώδικας δημιουργεί ένα διάνυσμα με παραμέτρους x, y, z από 2 σημεία. Τα δύο αυτά σημεία μπορούν να καθοριστούν είτε μέσα από το Grasshopper είτε να δημιουργηθούν από το χρήστη χρησιμοποιώντας κώδικα (δες Params->Geometry->Point).

Μεταβλητές εισόδου:

ptA – Πρώτο σημείο ptB – Δεύτερο σημείο

Μεταβλητές εξόδου:

Vector – Διάνυσμα που δημιουργείται VectorMagnitude – Μέτρο του διανύσματος

```
import rhinoscriptsyntax as rs
import ghpythonlib.components as ghc

Vector = ghc.Vector2Pt(ptA, ptB).vector
VectorMagnitude = ghc.VectorLength(Vector)
```

Curve:

Analysis:

End Points:

Ο παρακάτω κώδικας δέχεται σαν είσοδο μια ανοιχτή καμπύλη και επιστρέφει ως σημεία τα άκρα της.

Μεταβλητές εισόδου:

curve – Καμπύλη

Μεταβλητές εξόδου:

StartPoint – Σημείο αρχής καμπύλης EndPoint – Σημείο τέλους καμπύλης

```
import ghpythonlib.components as ghc

StartPoint = ghc.EndPoints(curve).start
EndPoint = ghc.EndPoints(curve).end
```

Evaluate Curve:

Ο παρακάτω κώδικας δέχεται σαν είσοδο μια καμπύλη και μια παράμετρο (κατά μήκος της καμπύλης) σύμφωνα με την οποία επιστρέφει πληροφορίες της καμπύλης: το σημείο της καμπύλης στην παράμετρο, την εφαπτομένη της καμπύλης στη παράμετρο και γωνία μεταξύ της εισερχόμενης και εξερχόμενης καμπύλης στη παράμετρο.

Μεταβλητές εισόδου:

curve – Καμπύλη parameter - Παράμετρος

Μεταβλητές εξόδου:

Point – Το σημείο της καμπύλης στην παράμετρο
Tangent – Εφαπτομένη της καμπύλης στη παράμετρο
Angle – Η γωνία μεταξύ της εισερχόμενης και εξερχόμενης καμπύλης στη παράμετρο

```
import ghpythonlib.components as ghc

Point = ghc.EvaluateCurve(curve, parameter).point
Tangent = ghc.EvaluateCurve(curve, parameter).tangent
```

```
Angle = ghc.EvaluateCurve(curve, parameter).angle
```

Division:

Divide Curve:

Ο παρακάτω κώδικας δέχεται σαν είσοδο μια καμπύλη, στα πόσα κομμάτια θέλουμε να τη διαιρέσουμε και αν επιθυμούμε να τη διαιρέσουμε στα σημεία που γίνεται πολύ απότομη. Σαν έξοδο δίνει τα σημεία στα οποία διαιρέθηκε η καμπύλη, τις εφαπτομένες της καμπύλης στα σημεία όπου έγινε διαίρεση και οι κατά μήκος της καμπύλης αποστάσεις των σημείων αυτών από την αρχή της καμπύλης.

Μεταβλητές εισόδου:

curve – Καμπύλη segmentsNum – Αριθμός επιθυμητών υποδιαιρέσεων
divideAtKinks – Μεταβλητή τύπου Boolean για να ρυθμίσουμε τη διαίρεση στα απότομα σημεία της καμπύλης

Μεταβλητές εξόδου:

Points – Τα σημεία όπου έγινε διαίρεση
Tangents – Οι εφαπτομένες της καμπύλης στα σημεία διαίρεσης
Parameters – Οι αποστάσεις των σημείων διαίρεσης από την αρχή της καμπύλης

```
import ghpythonlib.components as ghc

'''Αντικαταστήστε τα segmentsNum και divideAtKinks πιο κάτω για να προκαθορίσετε τις τιμές τους'''

Points = ghc.DivideCurve(curve, segmentsNum, divideAtKinks).points
Tangents = ghc.DivideCurve(curve, segmentsNum, divideAtKinks).tangents
Parameters = ghc.DivideCurve(curve, segmentsNum, divideAtKinks).parameters
```

Contour:

Ο παρακάτω κώδικας δέχεται ως είσοδο μια λίστα από καμπύλες, ένα σημείο, ένα διάνυσμα κατεύθυνσης και έναν αριθμό για την επιθυμητή απόσταση μεταξύ των ισοϋψών καμπύλων που δημιουργούνται με βάση αυτά τα στοιχεία.

Μεταβλητές εισόδου:

curve – Λίστα καμπύλων point – Σημείο direction – Διάνυσμα κατεύθυνσης
distance – Απόσταση μεταξύ ισοϋψών καμπύλων

Μεταβλητές εξόδου:

Contours – Ισοϋψείς καμπύλες Parameters – Παράμετροι των ισοϋψών καμπύλων

```
import ghpythonlib.components as ghc

Contours = ghc.Contour(curve, point, direction, distance).contours
Parameters = ghc.Contour(curve, point, direction, distance).parameters
```

Primitive:

Circle (από ακτίνα και επίπεδο):

Ο παρακάτω κώδικας δημιουργεί ένα κύκλο με δεδομένα την ακτίνα και το επίπεδο (plane). Η ακτίνα μπορεί να οριστεί από άλλα μέλη του Grasshopper (sliders, control knobs κλπ) ή να οριστεί από τον χρήστη μέσα στον κώδικα του. Παρομοίως το επίπεδο μπορεί να επιλεγεί μέσα από το περιβάλλον του Grasshopper ή να δημιουργηθεί μέσω κώδικα από το χρήστη (δες Params->Geometry->Plane). ***Σημείωση:*** αν δεν δοθεί επίπεδο θεωρείται πως είναι το επίπεδο xy.

Μεταβλητές εισόδου:

Plane – Επίπεδο Radius – Ακτίνα

Μεταβλητές εξόδου:

Circle – Κύκλος που δημιουργείται

```
import ghpythonlib.components as ghc

'''Αφαιρέστε το # από τις παρακάτω γραμμές για να προκαθορίσετε τη τιμή της ακτίνας'''

#Radius = 1

Circle = ghc.Circle(Plane, Radius)
```

Circle (από τρία σημεία):

Ο παρακάτω κώδικας δημιουργεί ένα κύκλο με δεδομένα τρία σημεία. Τα σημεία μπορούν να επιλεγούν μέσα από το περιβάλλον του Grasshopper ή να δημιουργηθεί μέσω κώδικα από το χρήστη (δες Params->Geometry->Point).

Μεταβλητές εισόδου:

PointA – Σημείο Α PointB – Σημείο Β PointC – Σημείο Γ

Μεταβλητές εξόδου:

Circle – Κύκλος που δημιουργείται μαζί με το επίπεδο και την ακτίνα του

```
import ghpythonlib.components as ghc

Circle = ghc.Circle3Pt(ptA, ptB, ptC)
```

Circle (από σημείο, διάνυσμα και ακτίνα):

Ο παρακάτω κώδικας δημιουργεί ένα κύκλο με δεδομένα ένα σημείο (το κέντρο του κύκλου), ένα διάνυσμα (το κανονικό διάνυσμα του επιπέδου) και την ακτίνα του κύκλου. Το σημείο του κέντρου μπορεί να επιλεγεί μέσα από το περιβάλλον του Grasshopper ή να δημιουργηθεί μέσω κώδικα από το χρήστη (δες Params->Geometry->Point). Η ακτίνα μπορεί να οριστεί από άλλα μέλη του

Grasshopper (sliders, control knobs κλπ) ή να οριστεί από τον χρήστη μέσα στον κώδικα του. Παρομοίως το διάνυσμα μπορεί να καθοριστεί μέσα από το περιβάλλον του Grasshopper ή να δημιουργηθεί μέσω κώδικα από το χρήστη (δες Params->Geometry->Vector).

Μεταβλητές εισόδου:

Center – Σημείο-Κέντρο του κύκλου Vector – Διάνυσμα Radius - Ακτίνα

Μεταβλητές εξόδου:

Circle – Κύκλος που δημιουργείται

```
import ghpythonlib.components as ghc

'''Αφαιρέστε το # από τις παρακάτω γραμμές για να προκαθορίσετε τη τιμή της ακτίνας'''

#Radius = 1

Circle = ghc.CircleCNR(Center, Vector, Radius)
```

Circle (που να ικανοποιεί πλήθος σημείων):

Ο παρακάτω κώδικας χρησιμοποιεί ως δεδομένο μια λίστα σημείων σύμφωνα με την οποία κατασκευάζει ένα κύκλο που περνά από τα σημεία αυτά.

Μεταβλητές εισόδου:

points – Λίστα σημείων

Μεταβλητές εξόδου:

Circle – Κύκλος που δημιουργείται

```
import ghpythonlib.components as ghc

circle = ghc.CircleFit(points)
```

Circle (που εφάπτεται σε δύο καμπύλες):

Ο παρακάτω κώδικας δημιουργεί ένα κύκλο που εφάπτεται σε δύο καμπύλες. Οι καμπύλες μπορούν να οριστούν μέσω του Grasshopper ή να δημιουργηθούν μέσω κώδικα από το χρήστη. Το σημείο του κέντρου μπορεί να επιλεγεί μέσα από το περιβάλλον του Grasshopper ή να δημιουργηθεί μέσω κώδικα από το χρήστη (δες Params->Geometry->Point).

Μεταβλητές εισόδου:

CurveA – Καμπύλη Α CurveB – Καμπύλη Β Center – Σημείο κέντρου κύκλου

Μεταβλητές εξόδου:

Circle – Κύκλος που δημιουργείται

```
import ghpythonlib.components as ghc

Circle = ghc.CircleTanTan(CurveA, CurveB, Center)
```

Circle (που εφάπτεται σε τρείς καμπύλες):

Ο παρακάτω κώδικας δημιουργεί ένα κύκλο που εφάπτεται σε τρείς καμπύλες. Οι καμπύλες μπορούν να οριστούν μέσω του Grasshopper ή να δημιουργηθούν μέσω κώδικα από το χρήστη. Το σημείο του κέντρου μπορεί να επιλεγεί μέσα από το περιβάλλον του Grasshopper ή να δημιουργηθεί μέσω κώδικα από το χρήστη (δες Params->Geometry->Point).

Μεταβλητές εισόδου:

CurveA – Καμπύλη Α CurveB – Καμπύλη Β CurveC – Καμπύλη Γ

Center – Σημείο κέντρου κύκλου

Μεταβλητές εξόδου:

Circle – Κύκλος που δημιουργείται

```
import ghpythonlib.components as ghc

Circle = ghc.CircleTanTanTan(CurveA, CurveB, CurveC, Center)
```

Arc:

Arc (από επίπεδο, ακτίνα και γωνία):

Ο παρακάτω κώδικας δημιουργεί ένα τόξο με δεδομένα το επίπεδο, την ακτίνα και τη γωνία. Το επίπεδο μπορεί να καθοριστεί μέσω του Grasshopper ή μέσω κώδικα. Η ακτίνα μπορεί να οριστεί από άλλα μέλη του Grasshopper (sliders, control knobs κλπ) ή να οριστεί από τον χρήστη μέσα στον κώδικα του. Παρομοίως το επίπεδο μπορεί να επιλεγεί μέσα από το περιβάλλον του Grasshopper ή να δημιουργηθεί μέσω κώδικα από το χρήστη (δες Params->Geometry->Plane).

Μεταβλητές εισόδου:

Plane – Επίπεδο Radius – Ακτίνα Angle - Γωνία

Μεταβλητές εξόδου:

Arc – Τόξο που δημιουργείται

```
import rhinoscriptsyntax as rs
import ghpythonlib.components as ghc

#Το Angle είναι σε ακτίνια
#Αν θέλουμε η είσοδος της γωνίας μας να είναι σε μοίρες,
#στην επόμενη γραμμή αντικαθιστούμε το Angle με Angle*0.0175
```

```
AngleDomain = ghc.ConstructDomain(0, AngleDomain)
Arc = ghc.Arc(Plane, Radius, Angle)
```

Arc (από αρχικό, ενδιάμεσο και τελικό σημείο):

Ο παρακάτω κώδικας δημιουργεί ένα τόξο με δεδομένα το αρχικό, ενδιάμεσο και τελικό σημείο. Τα σημεία αυτά μπορούν να επιλεγούν μέσα από το περιβάλλον του Grasshopper ή να δημιουργηθούν μέσω κώδικα από το χρήστη (δες Params->Geometry->Point).

Μεταβλητές εισόδου:

StartPt – Αρχικό σημείο InteriorPt – Ενδιάμεσο σημείο EndPt –Τελικό σημείο

Μεταβλητές εξόδου:

Arc – Τόξο που δημιουργείται Plane – Το επίπεδο του τόξου

Radius – Η ακτίνα του τόξου

```
import rhinoscriptsyntax as rs
import ghpythonlib.components as ghc

'''Αφαιρέστε το # από τις παρακάτω γραμμές για να προκαθορίσετε τις τιμές του
αρχικού, ενδιάμεσου και τελικού σημείου της καμπύλης'''
#StartPt = ghc.ConstructPoint(x1,y1,z1)
#InteriorPt = ghc.ConstructPoint(x2,y2,z2)
#EndPt = ghc.ConstructPoint(x3,y3,z3)

Arc = ghc.Arc3Pt(StartPt, InteriorPt, EndPt).arc
Plane = ghc.Arc3Pt(StartPt, InteriorPt, EndPt).plane
Radius = ghc.Arc3Pt(StartPt, InteriorPt, EndPt).radius
```

Arc (από αρχικό και τελικό σημείο και εφαπτόμενο διάνυσμα):

Ο παρακάτω κώδικας δημιουργεί ένα τόξο με δεδομένα το αρχικό και τελικό σημείο και ένα εφαπτόμενο διάνυσμα. Τα σημεία και το διάνυσμα αυτό μπορούν να επιλεγούν μέσα από το περιβάλλον του Grasshopper ή να δημιουργηθούν μέσω κώδικα από το χρήστη (δες Params->Geometry->Point και Params->Geometry->Vector).

Μεταβλητές εισόδου:

StartPt – Αρχικό σημείο EndPt –Τελικό σημείο Vector – Εφαπτόμενο διάνυσμα

Μεταβλητές εξόδου:

Arc – Τόξο που δημιουργείται Plane – Το επίπεδο του τόξου

Radius – Η ακτίνα του τόξου

```
import rhinoscriptsyntax as rs
import ghpythonlib.components as ghc
```

```
'''Αφαιρέστε το # από τις παρακάτω γραμμές για να προκαθορίσετε τις τιμές του αρχικού, ενδιάμεσου και τελικού σημείου της καμπύλης'''
#StartPt = ghc.ConstructPoint(x1,y1,z1)
#EndPt = ghc.ConstructPoint(x3,y3,z3)

Arc = ghc.ArcSED(StartPt, EndPt, Vector).arc
Plane = ghc.ArcSED(StartPt, EndPt, Vector).plane
Radius = ghc.ArcSED(StartPt, EndPt, Vector).radius
```

Spline:

Nurbs Curve:

Ο παρακάτω κώδικας δημιουργεί μια καμπύλη από μια λίστα σημείων, μια μεταβλητή που ρυθμίζει το βαθμό καμπύλωσης και μια μεταβλητή τύπου bool που ρυθμίζει το κατά πόσο η καμπύλη εξόδου είναι κλειστή ή όχι. Σαν έξοδο ο κώδικας αυτός έχει τη καμπύλη που δημιουργείται, το μήκος της και το πεδίο της.

Μεταβλητές εισόδου:

vertices – Λίστα σημείων degree – Μεταβλητή ρύθμισης «γωνίας» καμπύλης
periodic – Μεταβλητή bool που ρυθμίζει αν η καμπύλη είναι κλειστή

Μεταβλητές εξόδου:

Curve – Καμπύλη Length – Μήκος καμπύλης Domain – Πεδίο καμπυλης

```
import ghpythonlib.components as ghc

'''Αλλάξτε τα degree και periodic πιο κάτω αν θέλετε να προκαθορίσετε τις τιμές τους'''

Curve = ghc.NurbsCurve(vertices, degree, periodic).curve
Length = ghc.NurbsCurve(vertices, degree, periodic).length
Domain = ghc.NurbsCurve(vertices, degree, periodic).domain
```

Interpolate:

Ο παρακάτω κώδικας δημιουργεί μια καμπύλη που περνά από μια λίστα σημείων, μια μεταβλητή που ρυθμίζει το βαθμό καμπύλωσης, μια μεταβλητή τύπου bool που ρυθμίζει το κατά πόσο η καμπύλη εξόδου είναι κλειστή ή όχι και μια μεταβλητή που καθορίζει το τύπο του κόμβου. Σαν έξοδο ο κώδικας αυτός έχει τη καμπύλη που δημιουργείται, το μήκος της και το πεδίο της.

Μεταβλητές εισόδου:

vertices – Λίστα σημείων degree – Μεταβλητή ρύθμισης «γωνίας» καμπύλης
periodic – Μεταβλητή bool που ρυθμίζει αν η καμπύλη είναι κλειστή
knotStyle – Παίρνει τιμές από 0 μέχρι 2 ρυθμίζοντας το τύπο των κόμβων

Μεταβλητές εξόδου:

Curve – Καμπύλη Length – Μήκος καμπύλης Domain – Πεδίο καμπυλης

```
import ghpythonlib.components as ghc
```

```
'''Αλλάξτε τα degrees, periodic και knotStyle πιο κάτω αν επιθυμείτε να
προκαθορίσετε τις τιμές τους'''

Curve = ghc.Interpolate(vertices, degrees, periodic, knotStyle).curve
Length = ghc.Interpolate(vertices, degrees, periodic, knotStyle).length
Domain = ghc.Interpolate(vertices, degrees, periodic, knotStyle).domain
```

Util:

Explode:

Ο παρακάτω κώδικας δέχεται σαν είσοδο μια καμπύλη και μια μεταβλητή τύπου bool που ρυθμίζει αν θα «σπάζει» η καμπύλη επανειλημμένα. Σαν έξοδο ο κώδικας αυτός έχει τα κομμάτια και τα σημεία όπου σπάει η καμπύλη.

Μεταβλητές εισόδου:

curve – Καμπύλη
recursive – Μεταβλητή που ρυθμίζει αν η καμπύλη θα σπάζει επανειλημμένα

Μεταβλητές εξόδου:

Segments – Τα κομμάτια στα οποία σπάει η καμπύλη
Vertices – Τα σημεία όπου σπάει η καμπύλη

```
import ghpythonlib.components as ghc

'''Αντικαταστήστε το recursive πιο κάτω για να προκαθορίσετε τη τιμή του'''

Segments = ghc.Explode(curve, recursive).segments
Vertices = ghc.Explode(curve, recursive).vertices
```

Join Curves:

Ο παρακάτω κώδικας δέχεται σαν είσοδο μια λίστα από καμπύλες και μια μεταβλητή τύπου bool που ρυθμίζει αν η νέα καμπύλη, που θα προκύψει από την ένωση των καμπύλων της λίστας, θα διατηρήσει την φορά των καμπυλών που ενώθηκαν. Σαν έξοδο ο κώδικας αυτός έχει την καμπύλη που προέκυψε από την ένωση των καμπυλών της λίστας αλλά και τις καμπύλες που δεν ήταν δυνατόν να ενωθούν.

Μεταβλητές εισόδου:

curves – Λίστα καμπυλών προς ένωση
keepDirectoin – Μεταβλητή που ρυθμίζει αν νέα καμπύλη θα διατηρήσει τη φορά των αρχικών

Μεταβλητές εξόδου:

CurvesJoined – Λίστα που περιέχει τη καμπύλη της ένωσης αλλά και τις καμπύλες που δεν ήταν δυνατόν να ενωθούν

```
import ghpythonlib.components as ghc

'''Αντικαταστήστε το keepDirection πιο κάτω για να προκαθορίσετε τη τιμή του'''
```

```
CurvesJoined = ghc.JoinCurves(curvesIn, keepDirection)
```

Surface:

Analysis:

Deconstruct Brep:

Ο παρακάτω κώδικας δέχεται σαν είσοδο ένα στερεό (BREP) και σαν έξοδο δίνει σαν λίστες τις έδρες, τις ακμές και τις κορυφές του.

Μεταβλητές εισόδου:

brep - Στερεό

Μεταβλητές εξόδου:

Faces – Έδρες Edges – Ακμές Vertices - Κορυφές

```
import ghpythonlib.components as ghc

Faces = ghc.DeconstructBrep(brep).faces
Edges = ghc.DeconstructBrep(brep).edges
Vertices = ghc.DeconstructBrep(brep).vertices
```

Freeform:

4Point Surface:

Ο παρακάτω κώδικας δέχεται σαν είσοδο τέσσερα σημεία και σαν έξοδο δίνει μια επιφάνεια.

Μεταβλητές εισόδου:

pointA, pointB, pointC, pointD – Σημεία-άκρα επιφάνειας

Μεταβλητές εξόδου:

Surface – Επιφάνεια

```
import ghpythonlib.components as ghc

pointList = [pointA, pointB, pointD, pointC]
Surface = ghc.SurfaceFromPoints(pointList, 2, False)
```

Edge Surface:

Ο παρακάτω κώδικας δέχεται σαν είσοδο από δύο εώς τέσσερις καμπύλες και με βάση αυτές δημιουργεί μια επιφάνεια που δίνει σαν έξοδο.

Μεταβλητές εισόδου:

curveA, curveB, curveC, curveD – Καμπύλες εισόδου

Μεταβλητές εξόδου:

Surface – Επιφάνεια

```
import ghpythonlib.components as ghc

Surface = ghc.EdgeSurface(curveA, curveB, curveC, curveD)
```

Primitive:

Bounding Box:

Ο παρακάτω κώδικας δέχεται μια λίστα γεωμετρικών στοιχείων και ένα επίπεδο με βάση τα οποία κατασκευάζει ένα όσο το δυνατόν μικρότερο κουτί που να περιέχει τις γεωμετρίες αυτές. Σαν έξοδο ο κώδικας δίνει τα κουτιά που δημιουργεί σε συντεταγμένες χώρου και σε συντεταγμένες βασισμένες στο δοθέν επίπεδο.

Μεταβλητές εισόδου:

geometries – Λίστα γεωμετριών plane - Επίπεδο

Μεταβλητές εξόδου:

BoxWorldCoord – Λίστα των κουτιών σε συντεταγμένες χώρου
BoxPlaneCoord – Λίστα των κουτιών σε συντεταγμένες του δοθέν επιπέδου

```
import ghpythonlib.components as ghc

BoxWorldCoord = ghc.BoundingBox(geometries, plane).box
BoxPlaneCoord = ghc.BoundingBox(geometries, plane).box
```

Sphere:

Ο παρακάτω κώδικας παίρνει σαν δεδομένα ένα επίπεδο στο χώρο και έναν αριθμό σαν ακτίνα και δημιουργεί μια σφαίρα.

Μεταβλητές εισόδου:

plane – Επίπεδο radius - Ακτίνα

Μεταβλητές εξόδου:

Sphere - Σφαίρα

```
import ghpythonlib.components as ghc

'''Αντικαταστήστε το radius πιο κάτω για να προκαθορίσετε τη τιμή του'''

SphereOut = ghc.Sphere(plane, radius)
```

Util:

Divide Surface:

Ο παρακάτω κώδικας παίρνει σαν είσοδο μια επιφάνεια και τις επιθυμόμενες υποδιαιρέσεις στον u και v άξονα και δίνει σαν έξοδο τα σημεία όπου γίνεται η διαίρεση, τις συντεταγμένες αυτών των σημείων αλλά και τα κανονικά διανύσματα των σημείων όπου έγινε διαίρεση της επιφάνειας.

Μεταβλητές εισόδου:

surface – Επιφάνεια uAxisDiv – Υποδιαιρέσεις στο u άξονα
vAxisDiv – Υποδιαιρέσεις στο v άξονα

Μεταβλητές εξόδου:

Points – Σημεία όπου γίνεται υποδιάιρεση της επιφάνειας
NormVec – Λίστα των κανονικών διανυσμάτων των σημείων υποδιαίρεσης
Coordinates – Συντεταγμένες των σημείων υποδιαίρεσης

```
import ghpythonlib.components as ghc

'''Αντικαταστήστε τα uAxisDiv και yAxisDiv πιο κάτω για να προκαθορίσετε τις τιμές τους'''

Points = ghc.DivideSurface(surface, uAxisDiv, yAxisDiv).points
NormVec = ghc.DivideSurface(surface, uAxisDiv, yAxisDiv).normals
Coordinates = ghc.DivideSurface(surface, uAxisDiv, yAxisDiv).parameters
```

Isotrim:

Ο παρακάτω κώδικας δέχεται σαν είσοδο μια επιφάνεια και ένα πεδίο και σαν έξοδο δίνει ένα ισοπαραμετρικό υποσύνολο της επιφάνειας.

Μεταβλητές εισόδου:

surfaceIn - Επιφάνεια domain - Πεδίο

*Domain^2

Μεταβλητές εξόδου:

SurfaceOut – Ισοπαραμετρικό υποσύνολο επιφάνειας

```
import ghpythonlib.components as ghc

SurfaceOut = ghc.Isotrim(surfaceIn, domain)
```

Intersect:

Mathematical:

Curve | Line:

Ο παρακάτω κώδικας έχει ως είσοδο μια καμπύλη και μια ευθεία και δίνει σαν έξοδο τα σημεία τομής τους, τον αριθμό των σημείων τομής αλλά και τις αποστάσεις αυτών των σημείων από την αρχή της καμπύλης.

Μεταβλητές εισόδου:

curve – Καμπύλη line – Ευθεία

Μεταβλητές εξόδου:

Points – Λίστα των σημείων τομής PointsNum – Αριθμός σημείων τομής
Parameters – Λίστα αποστάσεων των σημείων τομής από αρχή της καμπύλης

```
import ghpythonlib.components as ghc

Points = ghc.CurveXLine(curve, line).points
Parameters = ghc.CurveXLine(curve, line).params
PointsNum = ghc.CurveXLine(curve, line).count
```

Physical:

Curve | Curve:

Ο παρακάτω κώδικας έχει ως είσοδο μια καμπύλη και μια ευθεία και δίνει σαν έξοδο τα σημεία τομής τους, τον αριθμό των σημείων τομής αλλά και τις αποστάσεις αυτών των σημείων από την αρχή της καμπύλης.

Μεταβλητές εισόδου:

curve – Καμπύλη line – Ευθεία

Μεταβλητές εξόδου:

Points – Λίστα των σημείων τομής PointsNum – Αριθμός σημείων τομής
Parameters – Λίστα αποστάσεων των σημείων τομής από αρχή της καμπύλης

```
import ghpythonlib.components as ghc

Points = ghc.CurveXCurve(curveA, curveB).points
ParamsA = ghc.CurveXCurve(curveA, curveB)[1]
ParamsB = ghc.CurveXCurve(curveA, curveB)[2]
```

Volume:

Ο παρακάτω κώδικας δέχεται σαν είσοδο ένα στερεό (BREP) και σαν έξοδο δίνει τον όγκο και το κέντρο μάζας του.

Μεταβλητές εισόδου:

brep - Στερεό

Μεταβλητές εξόδου:

Volume – Όγκος στερεού Centroid – Κέντρο μάζας στερεού

```
import ghpythonlib.components as ghc

Volume = ghc.Volume(brep).volume
Centroid = ghc.Volume(brep).centroid
```

Transform:

Affine:

Scale:

Ο παρακάτω κώδικας δέχεται σαν είσοδο ένα γεωμετρικό στοιχείο, ένα σημείο - κέντρο και μια μεταβλητή που ρυθμίζει τη μεγένθυση του γεωμτρικού στοιχείου. Το σημείο μπορεί να επιλεγεί μέσα από το περιβάλλον του Grasshopper ή να δημιουργηθεί μέσω κώδικα από το χρήστη (δες Params->Geometry->Point). Ο κώδικας αυτός σαν έξοδο δίνει το μεγενθυμένο γεωμετρικό στοιχείο αλλά και πληροφορίες για τη μεγένθυση του.

Μεταβλητές εισόδου:

geometryIn – Γεωμετρικό στοιχείο center – Σημείο κέντρου
scale – Παράγοντας μεγένθυσης

Μεταβλητές εξόδου:

GeometryOut – Μεγενθυμένη γεωμετρία
TransformData – Πληροφορίες για τη μεγένθυση

```
import ghpythonlib.components as ghc

'''Αλλάξτε το scale πιο κάτω για να προκαθορίσετε τη τιμή του'''

GeometryOut = ghc.Scale(geometryIn, center, scale).geometry
TransformData = ghc.Scale(geometryIn, center, scale).transform
```

Array:

Linear Array:

Ο παρακάτω κώδικας έχει σαν είσοδο ένα γεωμετρικό στοιχείο, ένα διάνυσμα και τον επιθυμητό αριθμό των στοιχείων του νέου διανύσματος. Σύμφωνα με αυτά δημιουργεί ένα διάνυσμα που αποτελείται από τον επιθυμητό αριθμό αυτού του γεωμετρικού στοιχείου προς την καθορισμένη κατεύθυνση. Σαν έξοδο έχει το νέο αυτό διάνυσμα από γεωμετρικά στοιχεία καθώς και πληροφορίες για το καθένα τους.

Μεταβλητές εισόδου:

geometryIn – Γεωμετρικό στοιχείο directionVec – Διάνυσμα κατεύθυνσης
numOfVecPar – Επιθυμητός αριθμός στοιχείων νέου διανύσματος

Μεταβλητές εξόδου:

GeometryOut – Διάνυσμα γεωμετρικών στοιχείων
TransformData – Πληροφορίες για τα στοιχεία του διανύσματος γεωμετρικών στοιχείων

```
import ghpythonlib.components as ghc

'''Αντικαταστήστε το numOfVecPar πιο κάτω για να προκαθορίσετε τη τιμή του'''

GeometryOut = ghc.LinearArray(geometryIn, directionVec, numOfVecPar).geometry
TransformData = ghc.LinearArray(geometryIn, directionVec, numOfVecPar).transform
```

Euclidean:

Move:

Ο παρακάτω κώδικας έχει σαν είσοδο ένα γεωμετρικό στοιχείο και ένα διάνυσμα μετατόπισης. Σύμφωνα με αυτά δημιουργεί ένα αντίγραφο του γεωμετρικού στοιχείου προς την καθορισμένη κατεύθυνση. Σαν έξοδο έχει το μετατοπισμένο γεωμετρικό στοιχείο καθώς και πληροφορίες για τη μετατόπιση του.

Μεταβλητές εισόδου:

geometryIn – Γεωμετρικό στοιχείο motionVec – Διάνυσμα μετατόπισης

Μεταβλητές εξόδου:

GeometryOut – Μετατοπισμένο γεωμετρικό στοιχείο
TransformData – Πληροφορίες για τη μετατόπιση του γεωμετρικού στοιχείου

```
import ghpythonlib.components as ghc

GeometryOut = ghc.Move(geometryIn, motionVec).geometry
TransformData = ghc.Move(geometryIn, motionVec).transform
```

Orient:

Ο παρακάτω κώδικας έχει σαν είσοδο ένα γεωμετρικό στοιχείο και δύο επίπεδα (ένα αρχικό και ένα τελικό). Σύμφωνα με αυτά τα στοιχεία, μεταφέρει τη καμπύλη από το αρχικό στο τελικό επίπεδο. Σαν έξοδο έχει το μεταφερμένο γεωμετρικό στοιχείο καθώς και πληροφορίες για τη μετακίνηση του.

Μεταβλητές εισόδου:

geometryIn – Γεωμετρικό στοιχείο source, target – Αρχικό και τελικό επίπεδο

Μεταβλητές εξόδου:

GeometryOut – Μεταφερμένο γεωμετρικό στοιχείο
TransformData – Πληροφορίες για τη μεταφορά του γεωμετρικού στοιχείου

```
import ghpythonlib.components as ghc

GeometryOut = ghc.Orient(geometryIn, source, target).geometry
TransformData = ghc.Orient(geometryIn, source, target).transform
```

Rotate 3D:

Ο παρακάτω κώδικας δέεται σαν είσοδο ένα γεωμετρικό στοιχείο, ένα σημείο - κέντρο, μια γωνία και έναν άξονα. Με αυτά τα στοιχεία, περιστρέφει το γεωμετρικό αυτό στοιχείο στην επιθυμητή γωνία γύρω από τον καθορισμένο άξονα. Σαν έξοδο έχει το μετατοπισμένο γεωμετρικό στοιχείο καθώς και πληροφορίες για τη μετατόπιση του.

Μεταβλητές εισόδου:

geometryIn – Γεωμετρικό στοιχείο
angle – Γωνία περιστροφής
axis – Διάνυσμα άξονα
center – Σημείο - κέντρο

Μεταβλητές εξόδου:

GeometryOut – Μετατοπισμένο γεωμετρικό στοιχείο
TransformData – Πληροφορίες για τη μετατόπιση του γεωμετρικού στοιχείου

```
import ghpythonlib.components as ghc

GeometryOut = ghc.Rotate3D(geometryIn, angle, center, axis).geometry
TransformData = ghc.Rotate3D(geometryIn, angle, center, axis).transform
```

Morph:

Box Morph:

Ο παρακάτω κώδικας δέχεται σαν είσοδο ένα γεωμετρικό στοιχείο, και δύο κουτιά (ένα αναφοράς και ένα ως στόχος). Με αυτά τα δεδομένα ο κώδικας μορφοποιεί το γεωμετρικό στοιχείο σε κουτί το οποίο δίνει σαν έξοδο.

Μεταβλητές εισόδου:

geometryIn – Γεωμετρικό στοιχείο
reference, target – Κουτιά αναφοράς κα στόχου

Μεταβλητές εξόδου:

GeometryOut – Κουτί στην έξοδο

```
import ghpythonlib.components as ghc

GeometryOut = ghc.BoxMorph(geometryIn, reference, target)
```

Surface Box:

Ο παρακάτω κώδικας δέχεται σαν είσοδο μια επιφάνεια, ένα πεδίο και μια μεταβλητή που ρυθμίζει το ύψος. Χρησιμοποιώντας αυτές τις πληροφορίες, κατασκευάζεται ένα κουτί που δίνεται σαν έξοδος.

Μεταβλητές εισόδου:

surface – Επιφάνεια
domain – Πεδίο
height - Ύψος

Μεταβλητές εξόδου:

Box - Κουτί

```
import ghpythonlib.components as ghc

Box = ghc.SurfaceBox(surface, domain, height)
```

Παραδείγματα:

Finger joints example

Ο παρακάτω κώδικας χρησιμοποιείται για τον καθορισμό μιας εγκοπής στο ένα από δυο συμπληρωματικά κομμάτια υλικού με σκοπό την μεταξύ τους ένωση. Σαν μεταβλητές εισόδου δέχεται μια καμπύλη, μια σειρά μεταβλητών που έχουν να κάνουν με την ανοικοδόμηση της καμπύλης (rebuild curve), μια μεταβλητή που ρυθμίζει τον αριθμό των εγκοπών, μια μεταβλητή τύπου bool που ρυθμίζει την κατεύθυνση των εγκοπών και μια μεταβλητή που καθορίζει το πάχος του υλικού και αντίστοιχα το μέγεθος των εγκοπών. Σαν έξοδος δίνεται σε μορφή συνεχόμενης καμπύλης η εγκοπή.

Μεταβλητές εισόδου:

curve – Καμπύλη ένωσης των δυο κομματιών
degree, count, tangents (rebuild curve) – Μεταβλητές ανοικοδόμησης καμπύλης
divides – Μεταβλητή που ρυθμίζει τον αριθμό των εγκοπών
Boolean – μεταβλητή για τον καθορισμό της φοράς των εγκοπών
Material – Μεταβλητή για το πάχος του υλικού

Μεταβλητή εξόδου:

curveJoin – Εγκοπή σε μορφή συνεχούς καμπύλης

Κώδικας:

```
import rhinoscriptsyntax as rs
import ghpythonlib.components as ghc
import math

curveRebuild=ghc.RebuildCurve(curve1,degree,count,tangents)

lengthCurve1=ghc.Length(curveRebuild)

lengthCurve1Div=ghc.Division(lengthCurve1,divides)

pattern=ghc.Division(lengthCurve1Div,2)
curve1DashPat=ghc.DashPattern(curveRebuild,pattern)
curveDash=ghc.ListItem(curve1DashPat,0,False)
curveGap=ghc.ListItem(curve1DashPat,1,False)

if (boolean==True):
    disp=-1.0
elif (boolean==False):
    disp=1.0

materialThick=ghc.Multiplication(disp,material)

p0=ghc.ConstructPoint(1000,1000)
p1=ghc.ConstructPoint(-1000,1000)
p2=ghc.ConstructPoint(-1000,-1000)
p3=ghc.ConstructPoint(1000,-1000)
surface_1=ghc.x4PointSurface(p0,p1,p2,p3)

curveOffset=ghc.OffsetonSrf(curveDash,materialThick,surface_1)
```

```
curveDash_start=ghc.EndPoints(curveDash).start
curveDash_end=ghc.EndPoints(curveDash).end

curveOffset_start=ghc.EndPoints(curveOffset).start
curveOffset_end=ghc.EndPoints(curveOffset).end

curveCon_start=ghc.Line(curveDash_start,curveOffset_start)
curveCon_end=ghc.Line(curveDash_end,curveOffset_end)

curveAll_a=ghc.Merge(curveGap,curveCon_start)
curveAll_b=ghc.Merge(curveOffset,curveCon_end)

curveAll=ghc.Merge(curveAll_a,curveAll_b)
curveJoin=ghc.JoinCurves(curveAll,False)
```

Πακετάρισμα:

Δημιουργία Grasshopper Component τύπου GHPY:

Η διαδικασία αυτή δημιουργεί ένα Grasshopper Component από κώδικα Python σαν αυτόν πιο πάνω χρησιμοποιώντας τον ενσωματομένο επεξεργαστή κώδικα Python για το Grasshopper.

Σημειώνεται πως αυτή η μέθοδος δημιουργίας component είναι δυνατή μόνο από το Rhino 6 και άνω (το οποίο αυτή τη στιγμή βρίσκεται σε δημόσια ανοιχτό δοκιμαστικό στάδιο WIP).

Πρώτο στάδιο στη δημιουργία του component είναι η συγγραφή του κώδικα και η διαμόρφωση των εισόδων / εξόδων του component κατά τα πρότυπα που δείχνει ο Ίκαρος πιο πάνω.

Στη συνέχεια προκειμένου να μπορεί ο κώδικας αυτός να μετατραπεί σε Grasshopper component, χρειάζεται να κάνουμε κάποιες μετατροπές και προσθήκες σε αυτόν.

Αρχικά, στο εντελώς πάνω μέρος του κώδικα, ανοίγουμε τριπλά εισαγωγικά (`"""`), μετά γράφουμε τις συνοπτικές οδηγίες χρήσης του component (docstrings), περιγράφουμε δηλαδή τις εισόδους και τις εξόδους του. Αυτό θα είναι το μήνυμα που θα βλέπει ο χρήστης όταν αφήνει το mouse πάνω από το component. Αφού τελειώσουμε με τη συγγραφή του μηνύματος χρήσης, κλείνουμε πάλι με τριπλά εισαγωγικά.

Στη συνέχεια, πρέπει στο κώδικα του component να εισάγουμε και το όνομα του συγγραφέα του, γι' αυτό ακριβώς κάτω από το μήνυμα χρήσης βάζουμε τη παρακάτω γραμμή αντικαθιστώντας το authorName με το όνομα του συγγραφέα.

```
__author__ = "authorName"
```

Σε αυτό το σημείο ο κώδικας μας πρέπει να έχει περίπου τη μορφή αυτή:

```
"""Σύντομη περιγραφή
      Είσοδοι:
            Περιγραφές εισόδων
      Έξοδοι:
            Περιγραφές εξόδων"""

__author__ = "authorName"
```

```
import ghpythonlib.components as ghc      #Βιβλιοθήκες που εισάγουμε

#Υπόλοιπος κώδικας του component
```

Το επόμενο βήμα είναι να αφαιρέσουμε όλα τα σχόλια (όχι όμως το μήνυμα χρήσης) από το κώδικα μας, καθότι αυτά δεν χρειάζονται.

Αφού τα έχουμε κάνει όλα αυτά, από τον επεξεργαστή κώδικα να επιλέξουμε να μπεί σε λειτουργία ανάπτυξης component (GH_Component SDK Mode). Την επιλογή αυτή βρίσκουμε στο μενού στο επάνω δεξί μέρος του επεξεργαστή κώδικα. Με την επιλογή αυτή, αυτόματα γίνεται κάποια μορφοποίηση του κώδικα και δε μένει παρά να κάνουμε κάποιες μικρές αλλαγές στο κώδικα για να βεβαιωθούμε πως τον κατανοεί καλά ο compiler.

Μορφοποιούμε το κώδικα ως εξής:

Docstrings
Εισαγωγή Βιβλιοθηκών
Όνομα συγγραφέα κώδικα
Κλάση component
    Βασική συνάρτηση RunScript
        Υπόλοιπος κώδικας

Σε αυτό το σημείο καλή ιδέα είναι (αν δεν το έχουμε κάνει ήδη), να σπάσουμε το πρόγραμμα μας σε υποσυναρτήσεις όπου αυτό είναι δυνατόν για να αυξηθούν οι επιδόσεις του component. Ο κώδικας μας θα είναι κάπως έτσι δηλαδή:

Κλάση component
    Βασική συνάρτηση RunScript
        Υπόλοιπος κώδικας
    Άλλες συναρτήσεις που καλούνται μέσα στη βασική συνάρτηση

Άλλο βασικό βήμα είναι να φροντίσουμε να έχουμε «δηλώσει» τις μεταβλητές μας πρωτού τις χρησιμοποιήσουμε, πράγμα που δεν είναι απαραίτητο κατά τη συγγραφή κώδικα Python αλλά απαραίτητο για την μετατροπή του κώδικα σε Grasshopper Component. Δηλαδή, έστω ότι στο πρόγραμμα μου χρησιμοποιώ μεταβλητές X,Y,Z όπου μερικές από αυτές μπορεί να είναι και εισόδου / εξόδου, στην αρχή της συνάρτησης τις δηλώνω ως εξής:

X = None Y = None Z = None

Το component μας χρειάζεται και ένα εικονίδιο, αυτό το σχεδιάζουμε σε ένα πρόγραμμα επεξεργασίας γραφικών (GIMP, Adobe Photoshop, MS Paint κλπ.) και το αποθηκεύουμε σαν αρχείο τύπου .png με διαστάσεις 24x24 pixel. Έπειτα τραβάμε το εικονίδιο που σχεδιάσαμε πάνω στο component.

Σε αυτό το σημείο μπορούμε να κάνουμε compile το component πηγαίνοντας πάλι στο μενού στα πάνω δεξιά του επεξεργαστή κώδικα και πατώντας την επιλογή Compile.

Στο παράθυρο που εμφανίζεται συμπληρώνουμε τα απαραίτητα στοιχεία για το component μας, σε ποια κατηγορία και υποκατηγορία θέλουμε να εμφανίζεται στο Grasshopper, και ένα κωδικό Guid σαν αυτό που εμφανίζεται πιο κάτω. Το κωδικό αυτό τον δημιουργεί τυχαία ο συγγραφέας

του component (αρκεί να έχει τη μορφή αυτή) ώστε το αν θελήσει να ενημερώσει το component του, δίνει στο καινούργιο τον ίδιο κωδικό και το Grasshopper να αναγνωρίζει το νέο component και αντικαθιστά το παλιό.

Πατώντας compile, ο compiler δημιουργεί το component μας και ζητά που να το αποθηκεύσει, αν το αποθηκεύσουμε στη προεπιλεγμένη τοποθεσία το Grasshopper θα το φορτώσει αυτόματα. Το αποτέλεσμα θα είναι ένα αρχείο python component για το Grasshopper (.ghpy).

Πληροφορίες για τη διαδικασία του πακεταρίσματος πάρθηκαν από την επίσημη ιστοσελίδα του Rhinoceros και μπορούν να βρεθούν εδώ:

https://discourse.mcneel.com/t/tutorial-creating-a-grasshopper-component-with-the-python-ghpy-compiler/38552

Το πρωτόκολλο Git και το Github:

Το τελευταίο κομμάτι αυτού του εγχειριδίου αποσκοπεί στο να κάνει μια σύντομη αλλά όσο το δυνατόν πιο πλήρη εισαγωγή στο πρωτόκολλο Git και σε μια πρακτική εφαρμογή του, το Github. Το πρωτόκολλο Git είναι ένα ανοιχτό σύστημα διαχείρισης εκδόσεων αρχείων (version control system – vcs) και οι πιο δημοφιλείς εφαρμογές του είναι στη διαχείριση κώδικα. Μια τέτοια εφαρμογή είναι και το Github. Οι γνώση της χρήσης τέτοιων εργαλείων θα κάνει τη συγγραφή, διανομή και βελτίωση κώδικα πολύ πιο εύκολη.

Για τη καλύτερη κατανόηση του Git αλλά και του Github, πρώτα θα δούμε κάποιες βασικές έννοιες του Git που μετέπειτα θα χρησιμοποιήσουμε σε ένα έμπρακτο παράδειγμα στο Github.

Βασικές έννοιες:

Repository (ή Repo):
Αποτελεί στην ουσία ένα φάκελο όπου μέσα τοποθετούνται ο κώδικας, τα σχετικά αρχεία και η τεκμιρίωση που έχουν σχέση με ένα πρόγραμμα. Ο φάκελος αυτός βρίσκεται στους servers του Github (στη προκειμένη περίπτωση) και μπορούμε να έχουμε τοπικό αντίγραφο του στον υπολογιστή μας.

Branch:
Ένα repository έχει ένα κεντρικό κλάδο (το master branch) και ο χρήστης μπορεί να δημιουργήσει όσα επιπλέον branches επιθυμεί. Ένα branch στην ουσία αποτελεί ένα παράλληλο αντίγραφο του βασικού (ή ενός άλλου υφιστάμενου) branch στο οποίο μπορούμε να κάνουμε όποια αλλαγή επιθυμούμε χωρίς να επηρεάζουμε το βασικό κλάδο. Τα branches συνήθως χρησιμοποιούνται για πειραματικές αλλαγές, δηλαδή ο χρήστης δημιουργεί έναν κλάδο, κάνει τις αλλαγές και βελτιώσεις που επιθυμεί σε αυτόν και μόλις είναι ικανοποιημένος με τις αλλαγές αυτές μπορεί, όπως θα δούμε και πιο κάτω, να τις περάσει αυτόματα και στον κεντρικό κλάδο (ή σε όποιον άλλο κλάδο επιθυμεί) αρκεί να είναι συμβατοί.

Forks:α fork είναι η δημιουργία ενός αντίγραφου ενός repository από άλλο χρήστη στο δικό μας λογαριασμό. Αυτό συνήθως γίνεται προκειμένου να κάνουμε αλλαγές και να τις προτείνουμε να ενταχθούν στο αυθεντικό repository ή για να δημιουργήσουμε εμείς μια δική μας έκδοση του αρχικού προγράμματος κάνοντας τις αλλαγές που επιθυμούμε.

Pull Requests (ή PR):
Έστω ότι έχουμε κάνει κάποια αλλαγή σε ένα branch και επιθυμούμε να τις εντάξουμε σε ένα άλλο branch είτε αυτό ανήκει στο δικό μας repository είτε αυτό ανήκει σε ένα αρχικό repository που το κάναμε fork. Προκειμένου να το κάνουμε αυτό δημιουργούμε το λεγόμενο Pull Request το οποίο στην ουσία αποτελεί ένα αίτημα για ενσωμάτωση των αλλαγών μας στο άλλο branch. Ένα PR συμπεριλαμβάνει την σειρά αλλαγών που πραγματοποιήσαμε καθώς και ένα τίτλο και μια περιγραφή για τις αλλαγές που κάναμε. Στην περίπτωση που το branch στο οποίο θέλουμε να ενταχθούν οι αλλαγές ανήκει σε εμάς (δηλαδή βρίσκεται κάτω από το δικό μας λογαριασμό), το PR μπορεί να γίνει αποδεκτό και να ενσωματωθεί (να γίνει merge δηλαδή) αυτόματα ή δίνοντας απλά την έγκριση μας. Αν όμως το branch στο οποίο θέλουμε να ενσωματώσουμε τις αλλαγές δεν ανήκει στο λογαριασμό μας, κάνουμε το PR και περιμένουμε την έγκριση του ιδιοκτήτη του branch να το εγκρίνει και να το κάνει merge. Για τέτοιες περιπτώσεις είναι που το PR έχει και δική του «σελίδα» στην οποία μπορούν να συνομιλήσουν και να προτείνουν αλλαγές όσοι τους αφορά το PR.